Transcript

1. Core Java Complete TutorialPrepared by:LOKESH KAKKARMail me at : kakkar.lokesh@gmail.comFollow me at : www.facebook.com/kakkar.lokeshCall me at : 9815-9900-67 1

2. Core Java TutorialSr. No. Topic Name01 OOPS Concepts02 Java Evolution03 Class Object basic04 Class Object Constructor overloading05 Inheritance06 Array and String07 Final Abstract class and interfaces08 Exceptions09 Streams10 GUI Applications11 Applet Programming12 Network Programming and Java Sockets13 Java Treads 2

3. Introduction to Object Oriented Design 3

4. Overview■ Understand Classes and Objects.■ Understand some of the key concepts/features in the Object Oriented paradigm.■ Benefits of Object Oriented Design paradigm. 4

5. OOP: model, map, reuse, extend ■ Model the real world problem to user's perceive; ■ Use similar metaphor in computational env. ■ Construct reusable components; ■ Create new components from existing ones. 5

6. Examples of Objects CAR BOY GIRL CLOCKVDU BOOK TREE TRIANGLE Figure 1.9: Examples of objects 6

7. Classes: Objects with the same attributes and behavior Person Objects Abstract Person Class Into Attributes: Name, Age, Sex Operations: Speak(), Listen(), Walk() Vehicle Objects Abstract Vehicle Class Into Attributes: Name, Model, Color Operations: Start(), Stop(), Accelerate() Polygon Objects Polygon Class Abstract Attributes: Vertices, Border, Into Color, FillColor Operations: Draw(), Erase(), Move() Figure 1.12: Objects and classes 7

8. Object Oriented Paradigm: Features Encapsulation Data Abstraction Single Inheritance Polymorphism OOP Paradigm Persistence Delegation Genericity Multiple Inheritance 8

9. Java's OO Features Encapsulation Data Abstraction Single Inheritance Polymorphism OOP JavaParadigm Persistence Delegation Genericity Multiple Inheritance 9

10. Encapsulation ■ It associates the Encapsulation code and the data it Data Abstraction manipulates into a single unit; and Single Inheritance keeps them safe from external interference Polymorphism OOP and misuse.Paradigm Persistence Delegation Genericity Data Functions Multiple Inheritance 10

11. Data Abstraction Encapsulation ■ The technique of creating new data types Data Abstraction that are well suited to an application. Single Inheritance ■ It allows the creation of user defined data types, Polymorphism OOP having the properties ofParadigm built data types and a set Persistence of permitted operators. Delegation ■ In Java, partial support. ■ In C++, fully supported Genericity (e.g., operator overloading). Multiple Inheritance 11

12. Abstract Data Type (ADT)■ A structure that contains both data and the actions to be performed on that data.■ Class is an implementation of an Abstract Data Type. 12

13. Class- Exampleclass Account { private String accountName; private double accountBalance; public withdraw(); public deposit(); public determineBalance();} // Class Account 13

14. Class■ Class is a set of attributes and operations that are performed on the attributes. Student Circle Account accountName name centre accountBalance age radius studentId withdraw() area() deposit() getName() circumference() determineBalance() getId() 14

15. Objects■ An Object Oriented system is a collection of interacting Objects.■ Object is an instance of a class. 15

16. Classes/Objects :John John and Jill areStudent objects of class Student :Jill :circleA circleA and circleBCircle are :circleB objects of class Circle 16

17. Class■ A class represents a template for several objects that have common properties.■ A class defines all the properties common to the object - attributes and methods.■ A class is sometimes called the object's type. 17

18. Object■ Objects have state and classes don't. John is an object (instance) of class Student. name = "John", age = 20, studentId = 1236 Jill is an object (instance) of class Student. name = "Jill", age = 22, studentId = 2345 circleA is an object (instance) of class Circle. centre = (20,10), radius = 25 circleB is an object (instance) of class Circle. centre = (0,0), radius = 10 18

19. Encapsulation■ All information (attributes and methods) in an object oriented system are stored within the object/class.■ Information can be manipulated through operations performed on the object/class – interface to the class. Implementation is hidden from the user.■ Object support Information Hiding – Some attributes and methods can be hidden from the user. 19

20. Encapsulation - Exampleclass Account { private String accountName; message private double accountBalance; Withdraw Deposit public withdraw(); public deposit(); message Account balance public determineBalance();} // Class Account Determine Balance message 20

21. Data Abstraction■ The technique of creating new data types that are well suited to an application.■ It allows the creation of user defined data types, having the properties of built in data types and more. 21

22. Abstraction - Exampleclass Account { private String accountName; Creates a data private double type Account accountBalance; public withdraw(); Account acctX; public deposit(); public determineBalance();} // Class Account 22

23. Inheritance■ New data types (classes) can be defined as extensions to previously defined types.■ Parent Class (Super Class) – Child Class (Sub Class)■ Subclass inherits Parent properties from the parent Inherited class. capability Child 23

24. Inheritance - Example■ Example ■ Define Person to be a class ■ A Person has attributes, such as age, height, gender ■ Assign values to attributes when describing object ■ Define student to be a subclass of Person ■ A student has all attributes of Person, plus attributes of his/her own (student no, course_enrolled) ■ A student has all attributes of Person, plus attributes of his/her own (student no, course_enrolled) ■ A student inherits all attributes of Person ■ Define lecturer to be a subclass of Person ■ Lecturer has all attributes of Person, plus attributes of his/her own (staff_id, subjectID1, subjectID2) 24

25. Inheritance - Example■ Circle Class can be a subclass (inherited from) of a parent class - Shape Shape Circle Rectangle 25

26. Inheritance - Example■ Inheritance can also have multiple levels. Shape Circle Rectangle GraphicCircle 26

27. Uses of Inheritance - Reuse■ If multiple classes have common attributes/methods, these methods can be moved to a common class - parent class.■ This allows reuse since the implementation is not repeated. Example : Rectangle and Circle method have a common method move(), which requires changing the centre coordinate. 27

28. Uses of Inheritance - Reuse Circle Rectangle centre centre height radius width area() area() circumference() circumference() move(newCentre) move(newCentre)move(newCentre){ centre = newCentre; move(newCentre){} centre = newCentre; } 28

29. Uses of Inheritance - Reuse Shape centre move(newCentre){ centre = newCentre area() } circumference() move(newCentre)Circle Rectangle heightradius widtharea() area()circumference() circumference() 29

30. Uses of Inheritance - Specialization■ Specialized behavior can be added to the child class.■ In this case the behaviour will be implemented in the child class. ■ E.g. The implementation of area() method in the Circle class is different from the Rectangle class.■ area() method in the child classes override the method in parent classes(). 30

31. Uses of Inheritance - Specialization Circle Rectangle centre centre height radius width area() area() circumference() circumference() move(newCentre) move(newCentre) area(){ return pi*r^2; area(){ } return height*width; } 31

32. Uses of Inheritance - Specialization Shape area(); - Not implemented centre And left for the child classes area() To implement circumference() move(newCentre) area(){Circle Rectangle return pi*r^2; } heightradius widtharea() area(){ area()circumference() return height*width; circumference() } 32

33. Uses of Inheritance – Common Interface■ All the operations that are supported for Rectangle and Circle are the same.■ Some methods have common implementation and others don't. ■ move() operation is common to classes and can be implemented in parent. ■ circumference(), area() operations are significantly different and have to be implemented in the respective classes.■ The Shape class

provides a common interface where all 3 operations move(), circumference() and area(). 33

34. Uses of Inheritance - Extension■ Extend functionality of a class.■ Child class adds new operations to the parent class but does not change the inherited behavior. ■ E.g. Rectangle class might have a special operation that may not be meaningful to the Circle class - rotate90degrees() 34

35. Uses of Inheritance - Extension Shape centre area() circumference() move(newCentre)Circle Rectangle heightradius width area()area() circumference()circumference() rotate90degrees() 35

36. Uses of Inheritance – Multiple Inheritance ■ Inherit properties from more than one class. ■ This is called Multiple Inheritance. Graphics Shape Circle 36

37. Uses of Multiple Inheritance■ This is required when a class has to inherit behavior from multiple classes.■ In the example Circle class can inherit move() operation from the Shape class and the paint() operation from the Graphics class.■ Multiple inheritance is not supported in JAVA but is supported in C++. 37

38. Uses of Inheritance – Multiple Inheritance GraphicCircle Shape color centre area() paint() circumference() move(newCentre) Circle radius area() circumference() 38

39. Polymorphism■ Polymorphic which means “many forms” has Greek roots. ■ Poly – many ■ Morphos - forms.■ In OO paradigm polymorphism has many forms.■ Allow a single object, method, operator associated with different meaning depending on the type of data passed to it. 39

40. Polymorphism■ An object of type Circle or Rectangle can be assigned to a Shape object. The behavior of the object will depend on the object passed. circleA = new Circle(); Create a new circle object Shape shape = circleA; shape.area(); area() method for circle class will be executed rectangleA = new Rectangle(); Create a new rectangle object shape= rectangle; shape.area() area() method for rectangle will be executed. 40

41. Polymorphism – Method Overloading■ Multiple methods can be defined with the same name, different input arguments. Method 1 - initialize(int a) Method 2 - initialize(int a, int b)■ Appropriate method will be called based on the input arguments. initialize(2) Method 1 will be called. initialize(2,4) Method 2 will be called. 41

42. Polymorphism – Operator Overloading ■ Allows regular operators such as +, -, *, / to have different meanings based on the type. ■ E.g. + operator for Circle can re-defined Circle c = c + 2; ■ Not supported in JAVA. C++ supports it. 42

43. Persistence■ The phenomenon where the object outlives the program execution.■ Databases support this feature.■ In Java, this can be supported if users explicitly build object persistency using IO streams. 43

44. Why OOP?■ Greater Reliability ■ Break complex software projects into small, self-contained, and modular objects■ Maintainability ■ Modular objects make locating bugs easier, with less impact on

the overall project■ Greater Productivity through Reuse!■ Faster Design and Modelling 44

45. Benefits of OOP..■ Inheritance: Elimination of Redundant Code and extend the use of existing classes.■ Build programs from existing working modules, rather than having to start from scratch. → save development time and get higher productivity.■ Encapsulation: Helps in building secure programs. 45

46. Benefits of OOP..■ Multiple objects to coexist without any interference.■ Easy to map objects in problem domain to those objects in the program.■ It is easy to partition the work in a project based on objects.■ The Data-Centered Design enables us in capturing more details of model in an implementable form. 46

47. Benefits of OOP..■ Object Oriented Systems can be easily upgraded from small to large systems.■ Message-Passing technique for communication between objects make the interface descriptions with external systems much simpler.■ Software complexity can be easily managed. 47

48. Summary■ Object Oriented Design, Analysis, and Programming is a Powerful paradigm■ Enables Easy Mapping of Real world Objects to Objects in the Program■ This is enabled by OO features: ■ Encapsulation ■ Data Abstraction ■ Inheritance ■ Polymorphism ■ Persistence■ Standard OO Design (UML) and Programming Languages (C++/Java) are readily accessible. 48

49. Java and its Evolution 49

50. Contents■ Java Introduction■ Java Features■ How Java Differs from other OO languages■ Java and the World Wide Web■ Java Environment■ Build your first Java Program■ Summary and Reference 50

51. Java - An Introduction■ Java - The new programming language developed by Sun Microsystems in 1991.■ Originally called Oak by James Gosling, one of the inventors of the Java Language.■ Java -The name that survived a patent search■ Java Authors: James , Arthur Van , and others■ Java is really “C++ -- ++ “ 51

52. Java Introduction■ Originally created for consumer electronics (TV, VCR, Freeze, Washing Machine, Mobile Phone).■ Java - CPU Independent language■ Internet and Web was just emerging, so Sun turned it into a language of Internet Programming.■ It allows you to publish a webpage with Java code in it. 52

53. Java MilestonesYear Development1990 Sun decided to developed special software that could be used for electronic devices. A project called Green Project created and head by James Gosling.1991 Explored possibility of using C++, with some updates announced a new language named “Oak”1992 The team demonstrated the application of their new language to control a list of home appliances using a hand held device. 53

54. Java MilestonesYear Development1994 The team developed a new Web browsed called “Hot Java” to locate and run Applets. HotJava gained instance success.1995 Oak was renamed to Java, as it

did not survive “legal” registration. Many companies such as Netscape and Microsoft announced their support for Java1996 Java established itself it self as both 1. “the language for Internet programming” 2. a general purpose OO language. 54

55. Sun white paper defines Java as: ■ Simple and Powerful ■ Safe ■ Object Oriented ■ Robust ■ Architecture Neutral and Portable ■ Interpreted and High Performance ■ Threaded ■ Dynamic 55

56. Java Attributes■ Familiar, Simple, Small■ Compiled and Interpreted■ Platform-Independent and Portable■ Object-Oriented■ Robust and Secure■ Distributed■ Multithreaded and Interactive■ High Performance■ Dynamic and Extensible 56

57. Java is Compiled and Interpreted Hardware and Programmer Operating System Source Code Byte CodeText Editor Compiler Interpreter .java file .class file java Notepad, java emacs,vi c appletviewer netscape 57

58. Compiled LanguagesProgrammer Source Code Object Executab Code le CodeText Editor Compiler linker .c file .o file a.out file Notepad, gcc emacs,vi 58

59. Total Platform Independence JAVA COMPILER (translator) JAVA BYTE CODE (same for all platforms) JAVA INTERPRETER (one for each different system)Windows 95 Macintosh Solaris Windows NT 59

60. Architecture Neutral & Portable■ Java Compiler - Java source code (file with extension .java) to bytecode (file with extension .class)■ Bytecode - an intermediate form, closer to machine representation■ A interpreter (virtual machine) on any target platform interprets the bytecode. 60

61. Architecture Neutral & Portable■ Porting the java system to any new platform involves writing an interpreter.■ The interpreter will figure out what the equivalent machine dependent code to run 61

62. Rich Class Environment■ Core Classes language Utilities Input/Output Low-Level Networking Abstract Graphical User Interface■ Internet Classes TCP/IP Networking WWW and HTML Distributed Programs 62

63. How Does Java Compares to C+ + and Other OO Languages 63

64. Overlap of C, C++, and Java C++ C Java 64

65. Java better than C++ ?■ No Typedefs, Defines, or Preprocessor■ No Global Variables■ No Goto statements ?■ No Pointers■ No Unsafe Structures■ No Multiple Inheritance■ No Operator Overloading■ No Automatic Coercions■ No Fragile Data Types 65

66. Object Oriented Languages -A Comparison Feature C++ Objective Ada Java CEncapsulation Yes Yes Yes YesInheritance Yes Yes No YesMultiple Inherit. Yes Yes No NoPolymorphism Yes Yes Yes YesBinding (Early or Late) Both Both Early LateConcurrency Poor Poor Difficult YesGarbage Collection No Yes No YesGenericity Yes No Yes LimitedClass Libraries Yes Yes Limited Yes 66

67. Java IntegratesPower of Compiled Languages and Flexibility of Interpreted Languages 67

68. Java Applications■ We can develop two types of Java programs: ■ Stand-alone applications ■ Web applications (applets) 68

69. Applications v/s Applets■ Different ways to run a Java executable are: Application- A stand-alone program that can be invoked from command line . A program that has a “main” method main Applet- A program embedded in a web page , to be run when the page is browsed . A program that contains no “main” method 69

70. Applets v/s Applications■ Different ways to run a Java executable are Application- A stand-alone program that can be invoked from command line . A program that has a “main” method main Applet- A program embedded in a web page , to be run when the page is browsed . A program that contains no “main” method■ Application –Executed by the Java interpreter.■ Applet- Java enabled web browser. 70

71. Java and World Wide Web Turning the Web into an Interactive and Application Delivery Platform 71

72. What is World Wide Web ?■ Web is an open-ended information retrieval system designed to be used in the Internet wide distributed system.■ It contains Web pages (created using HTML) that provide both information and controls.■ Unlike a menu driven system--where we are guided through a particular direction using a decision tree, the web system is open ended and we can navigate to a new document in any direction. 72

73. Web Structure of Information Search/Navigation 73

74. Execution of Applets 1 2 3 4 5 APPLET hello.class Create Accessing The browserDevelopment AT SUN’S Applet from creates“hello.java” WEB tag in Unimelb.edu.au a new AT SERVER HTML window and SUN.COM document a new thread and then runs the code Hello Java <app= “Hello”> The Internet Hello 74

75. Significance of downloading Applets ■ Interactive WWW ■ Flashy animation instead of static web pages ■ Applets react to users input and dynamically change ■ Display of dynamic data ■ WWW with Java - more than a document publishing medium ■ http://www.javasoft.com/applets/alpha/applets/Sto ckDemo/standalone.html 75

76. Power of Java and the Web■ Deliver applications, not just information■ Eliminate porting■ Eliminate end-user installation■ Slash software distribution costs■ Reach millions of customers - instantly 76

77. Java Development Kit■ javac - The Java Compiler■ java - The Java Interpreter■ jdb- The Java Debugger■ appletviewer -Tool to run the applets■ javap - to print the Java bytecodes■ javaprof - Java profiler■ javadoc - documentation generator■ javah - creates C header files 77

78. Java Environment 78

79. Java Development Kit■ javac - The Java Compiler■ java - The Java Interpreter■ jdb- The Java Debugger■ appletviewer -Tool to run the applets■ javap - to print the Java bytecodes■ javaprof - Java profiler■ javadoc - documentation generator■ javah - creates C header files 79

80. Process of Building and Running Java Programs Text Editor Java Source javadoc HTML Files Code javac Java Class File javah Header Files java jdb Outout 80

81. Let us Try OutBuilding your first Java Program 81

82. Hello Internet// hello.java: Hello Internet programclass HelloInternet{ public static void main(String args[]) { System.out.println("Hello Internet"); }} 82

83. Program Processing■ Compilation # javac hello.java results in HelloInternet.class■ Execution # java HelloInternet Hello Internet # 83

84. Simple Java Applet//HelloWorld.javaimport java.applet.Applet;import java.awt.*;public class HelloWorld extends Applet { public void paint(Graphics g) { g.drawString ("Hello World !",25, 25); }} 84

85. Calling an Applet<HTML><TITLE>HELLO WORLD APPLET</TITLE><HEAD>THE HELLO WORLD APPLET</HEAD><APPLET CODE="HelloWorld.class" width=500 height=500></APPLET></HTML> 85

86. Applet ExecutionUsing AppletViewer Using Browser 86

87. Summary■ Java has emerged as a general purpose OO language.■ It supports both stand alone and Internet Applications.■ Makes the Web Interactive and medium for application delivery.■ Provides an excellent set of Tools for Application Development.■ Java is ubiquitous! 87

88. Classes and Objects in Java Basics of Classes in Java 88

89. Contents■ Introduce to classes and objects in Java.■ Understand how some of the OO concepts learnt so far are supported in Java.■ Understand important features in Java classes. 89

90. Introduction■ Java is a true OO language and therefore the underlying structure of all Java programs is classes.■ Anything we wish to represent in Java must be encapsulated in a class that defines the "state" and "behaviour" of the basic program components known as objects.■ Classes create objects and objects use methods to communicate between them. They provide a convenient method for packaging a group of logically related data items and functions that work on them.■ A class essentially serves as a template for an object and behaves like a basic data type "int". It is therefore important to understand how the fields and methods are defined in a class and how they are used to build a Java program that incorporates the basic OO concepts such as encapsulation, inheritance, and polymorphism.90

91. Classes■ A class is a collection of fields (data) and methods (procedure or function) that operate on that data. Circle centre radius circumference() area() 91

92. Classes■ A class is a collection of fields (data) and methods (procedure or function) that operate on that data.■ The basic syntax for a class definition: class ClassName [extends SuperClassName] { [fields declaration] [methods declaration] }■ Bare bone class – no fields, no methods public class Circle { // my circle class } 92

93. Adding Fields: Class Circle with fields ■ Add fields public class Circle { public double x, y; // centre coordinate public double r; // radius of the circle } ■ The fields (data) are also called the instance varaibles. 93

94. Adding Methods■ A class with only data fields has no life. Objects created by such a class cannot respond to any messages.■ Methods are declared inside the body of the class but immediately after the declaration of data fields.■ The general form of a method declaration is: type MethodName (parameter-list) { Method-body; } 94

95. Adding Methods to Class Circle public class Circle { public double x, y; // centre of the circle public double r; // radius of circle //Methods to return circumference and area public double circumference() { return 2*3.14*r; } public double area() { Method Body return 3.14 * r * r; } } 95

96. Data Abstraction■ Declare the Circle class, have created a new data type – Data Abstraction■ Can define variables (objects) of that type: Circle aCircle; Circle bCircle; 96

97. Class of Circle cont. ■ aCircle, bCircle simply refers to a Circle object, not an object itself. aCircle bCircle null nullPoints to nothing (Null Reference) Points to nothing (Null Reference) 97

98. Creating objects of a class■ Objects are created dynamically using the new keyword.■ aCircle and bCircle refer to Circle objects aCircle = new Circle() ; bCircle = new Circle() ; 98

99. Creating objects of a classaCircle = new Circle();bCircle = new Circle() ;bCircle = aCircle; 99

100. Creating objects of a classaCircle = new Circle();bCircle = new Circle() ;bCircle = aCircle; Before Assignment Before Assignment aCircle bCircle aCircle bCircle P Q P Q 100

101. Automatic garbage collection Q■ The object does not have a reference and cannot be used in future.■ The object becomes a candidate for automatic garbage collection.■ Java automatically collects garbage periodically and releases the memory used to be used in the future. 101

102. Accessing Object/Circle Data■ Similar to C syntax for accessing data defined in a structure. ObjectName.VariableName ObjectName.MethodName(parameter-list) Circle aCircle = new Circle(); aCircle.x = 2.0 // initialize center and radius aCircle.y = 2.0 aCircle.r = 1.0 102

103. Executing Methods in Object/Circle■ Using Object Methods: sent ‘message’ to aCircle Circle aCircle = new Circle(); double area; aCircle.r = 1.0; area = aCircle.area(); 103

104. Using Circle Class// Circle.java: Contains both Circle class and its user class//Add Circle class code hereclass MyMain{ public static void main(String args[]) { Circle aCircle; // creating reference aCircle = new Circle(); // creating object aCircle.x = 10; // assigning value to data field aCircle.y = 20;

aCircle.r = 5; double area = aCircle.area(); // invoking method double circumf = aCircle.circumference(); System.out.println("Radius="+aCircle.r+" Area="+area); System.out.println("Radius="+aCircle.r+" Circumference ="+circumf); }} [raj@mundroo]%: java MyMain Radius=5.0 Area=78.5 Radius=5.0 Circumference =31.400000000000002 104

105. Summary■ Classes, objects, and methods are the basic components used in Java programming.■ We have discussed: ■ How to define a class ■ How to create objects ■ How to add data fields and methods to classes ■ How to access data fields and methods to classes 105

106. Classes and Objects in JavaConstructors, Overloading, Static Members 106

107. Refer to the Earlier Circle Program // Circle.java: Contains both Circle class and its user class //Add Circle class code here class MyMain { public static void main(String args[]) { Circle aCircle; // creating reference aCircle = new Circle(); // creating object aCircle.x = 10; // assigning value to data field aCircle.y = 20; aCircle.r = 5; double area = aCircle.area(); // invoking method double circumf = aCircle.circumference(); System.out.println("Radius="+aCircle.r+" Area="+area); System.out.println("Radius="+aCircle.r+" Circumference ="+circumf); } } [raj@mundroo]%: java MyMain Radius=5.0 Area=78.5 Radius=5.0 Circumference =31.400000000000002 107

108. Better way of Initialising or Access Data Members x, y, r ■ When there too many items to update/access and also to develop a readable code, generally it is done by defining specific method for each purpose. ■ To initialise/Update a value: ■ aCircle.setX(10) ■ To access a value: ■ aCircle.getX() ■ These methods are informally called as Accessors or Setters/Getters Methods. 108

109. Accessors – “Getters/Setters” public class Circle { public double x,y,r; //Methods to return circumference and area public double getX() { return x;} public double getY() { return y;} public double getR() { return r;} public double setX(double x_in) { x = x_in;} public double serY(double y_in) { y = y_in;} public double setR(double r_in) { r = r_in;} } 109

110. How does this code looks ? More readable ?// Circle.java: Contains both Circle class and its user class//Add Circle class code hereclass MyMain{ public static void main(String args[]) { Circle aCircle; // creating reference aCircle = new Circle(); // creating object aCircle.setX(10); aCircle.setY(20); aCircle.setR(5); double area = aCircle.area(); // invoking method double circumf = aCircle.circumference(); System.out.println("Radius="+aCircle.getR()+" Area="+area); System.out.println("Radius="+aCircle.getR()+" Circumference ="+circumf); }} [raj@mundroo]%: java MyMain Radius=5.0 Area=78.5 Radius=5.0 Circumference =31.400000000000002 110

111. Object Initialisation■ When objects are created, the initial value of data fields is unknown unless its users explicitly do so. For example, ■ ObjectName.DataField1 = 0; // OR ■ ObjectName.SetDataField1(0);■ In many cases, it makes sense if this initialisation can be carried out by default without the users explicitly initialising them. ■ For example, if you create an object of the class called “Counter”, it is natural to assume that the counter record- keeping field is initialised to zero unless otherwise specified differently. class Counter { int CounterIndex; … } Counter counter1 = new Counter(); ■ What is the value of “counter1.CounterIndex” ?■ In Java, this can be achieved though a

mechanism called constructors. 111

112. What is a Constructor?■ Constructor is a special method that gets invoked “automatically” at the time of object creation.■ Constructor is normally used for initializing objects with default values unless different values are supplied.■ Constructor has the same name as the class name.■ Constructor cannot return values.■ A class can have more than one constructor as long as they have different signature (i.e., different input arguments syntax). 112

113. Defining a Constructor■ Like any other method public class ClassName { // Data Fields… // Constructor public ClassName() { // Method Body Statements initialising Data Fields } //Methods to manipulate data fields }■ Invoking: ■ There is NO explicit invocation statement needed: When the object creation statement is executed, the constructor method will be executed automatically. 113

114. Defining a Constructor: Example public class Counter { int CounterIndex; // Constructor public Counter() { CounterIndex = 0; } //Methods to update or access counter public void increase() { CounterIndex = CounterIndex + 1; } public void decrease() { CounterIndex = CounterIndex - 1; } int getCounterIndex() { return CounterIndex; } } 114

115. Trace counter value at eachstatement and What is the output ? class MyClass { public static void main(String args[]) { Counter counter1 = new Counter(); counter1.increase(); int a = counter1.getCounterIndex(); counter1.increase(); int b = counter1.getCounterIndex(); if (a > b) counter1.increase(); else counter1.decrease(); System.out.println(counter1.getCounterIndex()); } } 115

116. A Counter with User Supplied Initial Value ? ■ This can be done by adding another constructor method to the class. public class Counter { int CounterIndex; // Constructor 1 public Counter() { CounterIndex = 0; } public Counter(int InitValue) { CounterIndex = InitValue; } } // A New User Class: Utilising both constructors Counter counter1 = new Counter(); Counter counter2 = new Counter (10); 116

117. Adding a Multiple-ParametersConstructor to our Circle Class public class Circle { public double x,y,r; // Constructor public Circle(double centreX, double centreY, double radius) { x = centreX; y = centreY; r = radius; } //Methods to return circumference and area public double circumference() { return 2*3.14*r; } public double area() { return 3.14 * r * r; } } 117

118. Constructors initialise Objects■ Recall the following OLD Code Segment: Circle aCircle = new Circle(); aCircle.x = 10.0; // initialize center and radius aCircle.y = 20.0 aCircle.r = 5.0; aCircle = new Circle() ; At creation time the center and radius are not defined. These values are explicitly set later. 118

119. Constructors initialise Objects■ With defined constructor Circle aCircle = new Circle(10.0, 20.0, 5.0); aCircle = new Circle(10.0, 20.0, 5.0) ; aCircle is created with center (10, 20) and radius 5 119

120. Multiple Constructors■ Sometimes want to initialize in a number of different ways, depending on circumstance.■ This can be supported by having multiple constructors having different input arguments. 120

121. Multiple Constructorspublic class Circle { public double x,y,r; //instance variables // Constructors public Circle(double centreX, double cenreY, double radius) { x = centreX; y = centreY; r = radius; } public Circle(double radius) { x=0; y=0; r = radius; } public Circle() { x=0; y=0; r=1.0; } //Methods to return circumference and area public double circumference() { return 2*3.14*r; } public double area() { return 3.14 * r * r; }} 121

122. Initializing with constructors public class TestCircles { public static void main(String args[]){ Circle circleA = new Circle(10.0, 12.0, 20.0); Circle circleB = new Circle(10.0); Circle circleC = new Circle(); } }circleA = new Circle(10, 12, 20) circleB = new Circle(10) circleC = new Circle() Centre = (10,12) Centre = (0,0) Radius = 20 Centre = (0,0) Radius = 1 Radius=10 122

123. Method Overloading■ Constructors all have the same name.■ Methods are distinguished by their signature: ■ name ■ number of arguments ■ type of arguments ■ position of arguments■ That means, a class can also have multiple usual methods with the same name.■ Not to confuse with method overriding (coming up), method overloading: 123

124. Polymorphism■ Allows a single method or operator associated with different meaning depending on the type of data passed to it. It can be realised through: ■ Method Overloading ■ Operator Overloading (Supported in C++, but not in Java)■ Defining the same method with different argument types (method overloading) - polymorphism.■ The method body can have different logic depending on the date type of arguments. 124

125. Scenario■ A Program needs to find a maximum of two numbers or Strings. Write a separate function for each operation. ■ In C: ■ int max_int(int a, int b) ■ int max_string(char *s1, char *s2) ■ max_int (10, 5) or max_string (“melbourne”, “sydney”) ■ In Java: ■ int max(int a, int b) ■ int max(String s1, String s2) ■ max(10, 5) or max(“melbourne”, “sydney”) ■ Which is better ? Readability and intuitive wise ? 125

126. A Program with Method Overloading// Compare.java: a class comparing different itemsclass Compare { static int max(int a, int b) { if(a > b) return a; else return b; } static String max(String a, String b) { if(a.compareTo (b) > 0) return a; else return b; } public static void main(String args[]) { String s1 = "Melbourne"; String s2 = "Sydney"; String s3 = "Adelaide"; int a = 10; int b = 20; System.out.println(max(a, b)); // which number is big System.out.println(max(s1, s2)); // which city is big System.out.println(max(s1, s3)); // which city is big }} 126

127. The New this keyword■ this keyword can be used to refer to the object itself. It is generally used for accessing class members (from its own methods) when they have the same name as those passed as arguments. public class Circle { public double x,y,r; // Constructor public Circle (double x, double y, double r) { this.x = x; this.y = y; this.r = r; } //Methods to return circumference and area } 127

128. Static Members■ Java supports definition of global methods and variables that can be accessed without creating objects of a class. Such members are called Static members.■ Define a variable by marking with the static methods.■ This feature is useful when we want to create a variable common to all instances of a class.■ One of the most common example is to have a variable that could

keep a count of how many objects of a class have been created.■ Note: Java creates only one copy for a static variable which can be used even if the class is never instantiated. 128

129. Static Variables■ Define using static: public class Circle { // class variable, one for the Circle class, how many circles public static int numCircles; //instance variables,one for each instance of a Circle public double x,y,r; // Constructors... }■ Access with the class name (ClassName.StatVarName): nCircles = Circle.numCircles; 129

130. Static Variables - Example■ Using static variables:public class Circle { // class variable, one for the Circle class, how many circles private static int numCircles = 0; private double x,y,r; // Constructors... Circle (double x, double y, double r){ this.x = x; this.y = y; this.r = r; numCircles++; }} 130

131. Class Variables - Example ■ Using static variables: public class CountCircles { public static void main(String args[]){ Circle circleA = new Circle(10, 12, 20); // numCircles = 1 Circle circleB = new Circle(5, 3, 10); // numCircles = 2 } }circleA = new Circle(10, 12, 20) circleB = new Circle(5, 3, 10) numCircles 131

132. Instance Vs Static Variables■ Instance variables : One copy per object. Every object has its own instance variable. ■ E.g. x, y, r (centre and radius in the circle)■ Static variables : One copy per class. ■ E.g. numCircles (total number of circle objects created) 132

133. Static Methods■ A class can have methods that are defined as static (e.g., main method).■ Static methods can be accessed without using objects. Also, there is NO need to create objects.■ They are prefixed with keyword “static”■ Static methods are generally used to group related library functions that don’t depend on data members of its class. For example, Math library functions. 133

134. Comparator class with Static methods // Comparator.java: A class with static data items comparision methods class Comparator { public static int max(int a, int b) { if(a > b) return a; else return b; } public static String max(String a, String b) { if(a.compareTo (b) > 0) return a; else return b; } } class MyClass { public static void main(String args[]) Directly accessed using ClassName (NO Objects) { String s1 = "Melbourne"; String s2 = "Sydney"; String s3 = "Adelaide"; int a = 10; int b = 20; System.out.println(Comparator.max(a, b)); // which number is big System.out.println(Comparator.max(s1, s2)); // which city is big System.out.println(Comparator.max(s1, s3)); // which city is big } } 134

135. Static methods restrictions■ They can only call other static methods.■ They can only access static data.■ They cannot refer to “this” or “super” (more later) in anyway. 135

136. Summary■ Constructors allow seamless initialization of objects.■ Classes can have multiple methods with the same name [Overloading]■ Classes can have static members, which serve as global members of all objects of a class.■ Keywords: constructors, polymorphism, method overloading, this, static variables, static methods. 136

137. InheritanceClasses and Subclasses Or Extending a Class (only for reusability) 137

138. Inheritance: Introduction■ Reusability--building new components by utilising existing components- is yet another important aspect of OO paradigm.■ It is always good/“productive” if we are

able to reuse something that is already exists rather than creating the same all over again.■ This is achieve by creating new classes, reusing the properties of existing classes. 138

139. Inheritance: Introduction■ This mechanism of deriving a new class from existing/old class is called “inheritance”. Parent■ The old class is known as “base” class, “super” Inherited capability class or “parent” class”; and the new class is known as “sub” class, Child “derived” class, or “child” class. 139

140. Inheritance: Introduction■ The inheritance allows subclasses to inherit all properties (variables and methods) of their parent classes. The different forms of inheritance are: ■ Single inheritance (only one super class) ■ Multiple inheritance (several super classes) ■ Hierarchical inheritance (one super class, many sub classes) ■ Multi-Level inheritance (derived from a derived class) ■ Hybrid inheritance (more than two types) ■ Multi-path inheritance (inheritance of some properties from two sources). 140

141. Forms of Inheritance A A B A B C B C D(a) Single Inheritance (b) Multiple Inheritance (c) Hierarchical Inheritance A A A B c B c B C D D(a) Multi-Level Inheritance (b) Hybrid Inheritance (b) Multipath Inheritance 141

142. Defining a Sub class■ A subclass/child class is defined as follows: class SubClassName extends SuperClassName { fields declaration; methods declaration; }■ The keyword “extends” signifies that the properties of super class are extended to the subclass. That means, subclass contains its own members as well of those of the super class. This kind of situation occurs when we want to enhance properties of existing class without actually modifying it. 142

143. Subclasses and Inheritance■ Circle class captures basic properties■ For drawing application, need a circle to draw itself on the screen, GraphicCircle...■ This can be realised either by updating the circle class itself (which is not a good Software Engineering method) or creating a new class that builds on the existing class and add additional properties. 143

144. Without Inheritance public class GraphicCircle { public Circle c; // keep a copy of a circle public double area() { return c.area(); } public double circumference (){ return c.circumference(); } // new instance variables, methods for this class public Color outline, fill; public void draw(DrawWindow dw) { /* drawing code here */ } }■ Not very elegant 144

145. Subclasses and Inheritance■ Circle class captures basic properties■ For drawing application need a circle to draw itself on the screen, GraphicCircle■ Java/OOP allows for Circle class code to be implicitly (re)used in defining a GraphicCircle■ GraphicCircle becomes a subclass of Circle, extending its capabilities 145

146. Subclassing Circle Circle x,y,r : double area ( ) : doubleSubclass, Derived circumference(): double Superclass class, or base class,Child class Or parent class GraphicCircle outline, fill : Color draw (DrawWindow) : void 146

147. Subclassing■ Subclasses created by the keyword extends: public class GraphicCircle extends Circle { // automatically inherit all the variables and methods // of Circle, so only need to put in the ‘new

stuff’ Color outline, fill; public void draw(DrawWindow dw) { dw.drawCircle(x,y,r,outline,fill); } }■ Each GraphicCircle object is also a Circle! 147

148. Final Classes■ Declaring class with final modifier prevents it being extended or subclassed.■ Allows compiler to optimize the invoking of methods of the class final class Cirlce{ …… } 148

149. Subclasses & Constructors■ Default constructor automatically calls constructor of the base class: default constructor for Circle class is called GraphicCircle drawableCircle = new GraphicCircle(); 149

150. Subclasses & Constructors■ Defined constructor can invoke base class constructor with super: public GraphicCircle(double x, double y, double r, Color outline, Color fill) { super(x, y, r); this.outline = outline; this fill = fill } 150

151. Shadowed Variables■ Subclasses defining variables with the same name as those in the superclass, shadow them: 151

152. Shadowed Variables - Examplepublic class Circle { public float r = 100;}public class GraphicCircle extends Circle { public float r = 10; // New variable, resolution in dots per inch}public class CircleTest { public static void main(String[] args){ GraphicCircle gc = new GraphicCircle(); Circle c = gc; System.out.println(“ GraphicCircleRadius= “ + gc.r); // 10 System.out.println (“ Circle Radius = “ + c.r); // 100 }} 152

153. Overriding Methods■ Derived/sub classes defining methods with same name, return type and arguments as those in the parent/super class, override their parents methods: 153

154. Overriding Methodsclass A { int j = 1; int f( ) { return j; }}class B extends A { int j = 2; int f( ) { return j; }} 154

155. Overriding Methodsclass override_test { public static void main(String args[]) { B b = new B(); System.out.println(b.j); // refers to B.j prints 2 System.out.println(b.f()); // refers to B.f prints 2 Object Type Casting A a = (A) b; System.out.println(a.j); // now refers to a.j prints 1 System.out.println(a.f()); // overridden method still refers to B.f() prints 2 ! } } [raj@mundroo] inheritance [1:167] java override_test 2 2 1 2 155

156. Using All in One: Person and Student Person name: String sex: char age: int Display ( ) : void Subclass class. Superclass class Student RollNo: int Branch: String Display() : void 156

157. Person class: Parent class// Student.java: Student inheriting properties of person classclass person{ private String name; protected char sex; // note protected public int age; person() { name = null; sex = U; // unknown age = 0; } person(String name, char sex, int age) { this.name = name; this.sex = sex; this.age = age; } String getName() { return name; } void Display() { System.out.println("Name = "+name); System.out.println("Sex = "+sex); System.out.println("Age = "+age); }} 157

158. Student class: Derived classclass student extends person{ private int RollNo; String branch;

student(String name, char sex, int age, int RollNo, String branch) { super(name, sex, age); // calls parent classs constructor with 3 arguments this.RollNo = RollNo; this.branch = branch; } void Display() // Method Overriding { System.out.println("Roll No = "+RollNo); System.out.println("Name = "+getName()); System.out.println("Sex = "+sex); System.out.println("Age = "+age); System.out.println("Branch = "+branch); } void TestMethod() // test what is valid to access { // name = "Mark"; Error: name is private sex = M; RollNo = 20; }} What happens if super class constructor is not explicitly invoked ? (default constructor will be invoked). 158

159. Driver Classclass MyTest{ public static void main(String args[]) { student s1 = new student("Rama", M, 21, 1, "Computer Science"); student s2 = new student("Sita", F, 19, 2, "Software Engineering"); System.out.println("Student 1 Details..."); s1.Display(); System.out.println("Student 2 Details..."); s2.Display(); person p1 = new person("Rao", M, 45); System.out.println("Person Details..."); p1.Display(); }} Can we create Object of person class ? 159

160. Output[raj@mundroo] inheritance [1:154] java MyTestStudent 1 Details...Roll No = 1Name = RamaSex = MAge = 21Branch = Computer ScienceStudent 2 Details...Roll No = 2Name = SitaSex = FAge = 19Branch = Software EngineeringPerson Details...Name = RaoSex = MAge = 45[raj@mundroo] inheritance [1:155] 160

161. Summary■ Inheritance promotes reusability by supporting the creation of new classes from existing classes.■ Various forms of inheritance can be realised in Java.■ Child class constructor can be directed to invoke selected constructor from parent using super keyword.■ Variables and Methods from parent classes can be overridden by redefining them in derived classes.■ New Keywords: extends, super, final 161

162. Arrays and Strings 162

163. Arrays - Introduction■ An array is a group of contiguous or related data items that share a common name.■ Used when programs have to handle large amount of data■ Each value is stored at a specific position■ Position is called a index or superscript. Base index = 0■ The ability to use a single name to represent a collection of items and refer to an item by specifying the item number enables us to develop concise and efficient programs. For example, a loop with index as the control variable can be used to read the entire array, perform calculations, and print out the results. 163

164. Arrays - Introduction 0 69 1 61index 2 70 3 89 values 4 23 5 10 6 9 164

165. Declaration of Arrays■ Like any other variables, arrays must declared and created before they can be used. Creation of arrays involve three steps: ■ Declare the array ■ Create storage area in primary memory. ■ Put values into the array (i.e., Memory location)■ Declaration of Arrays: ■ Form 1: Type arrayname[] ■ Form 2: ■ Type [] arrayname; ■ Examples: int[] students; int students[]; ■ Note: we don’t specify the size of arrays in the declaration. 165

166. Creation of Arrays■ After declaring arrays, we need to allocate memory for storage array items.■ In Java, this is carried out by using “new” operator, as follows: ■ Arrayname = new type[size];■

Examples: ■ students = new int[7]; 166

167. Initialisation of Arrays■ Once arrays are created, they need to be initialised with some values before access their content. A general form of initialisation is: ■ Arrayname [index/subscript] = value;■ Example: ■ students[0] = 50; ■ students[1] = 40;■ Like C, Java creates arrays starting with subscript 0 and ends with value one less than the size specified.■ Unlike C, Java protects arrays from overruns and under runs. Trying to access an array beyond its boundaries will generate an error message. 167

168. Arrays – Length■ Arrays are fixed length■ Length is specified at create time■ In java, all arrays store the allocated size in a variable named “length”.■ We can access the length of arrays as arrayName.length: e.g. int x = students.length; // x = 7■ Accessed using the index e.g. int x = students [1]; // x = 40 168

169. Arrays – Example// StudentArray.java: store integers in arrays and accesspublic class StudentArray{ public static void main(String[] args) { int[] students; students = new int[7]; System.out.println("Array Length = " + students.length); for (int i=0; i < students.length; i++) students[i] = 2*i; System.out.println("Values Stored in Array:"); for (int i=0; i < students.length; i++) System.out.println(students[i]); }}:w 169

170. Arrays – Initializing at Declaration ■ Arrays can also be initialised like standard variables at the time of their declaration. ■ Type arrayname[] = {list of values}; ■ Example: int[] students = {55, 69, 70, 30, 80}; ■ Creates and initializes the array of integers of length 5. ■ In this case it is not necessary to use the new operator. 170

171. Arrays – Example// StudentArray.java: store integers in arrays and accesspublic class StudentArray{ public static void main(String[] args) { int[] students = {55, 69, 70, 30, 80}; System.out.println("Array Length = " + students.length); System.out.println("Values Stored in Array:"); for (int i=0; i < students.length; i++) System.out.println(students[i]); }} 171

172. Two Dimensional Arrays Two dimensional Item1 Item2 Item3■ Sold arrays allows us to Person store data that are Salesgirl #1 10 15 30 recorded in table. For example: Salesgirl #2 14 30 33■ Table contains 12 items, we can think of Salesgirl #3 200 32 1 this as a matrix consisting of 4 rows Salesgirl #4 10 200 4 and 3 columns. 172

173. 2D arrays manipulations■ Declaration: ■ int myArray [][];■ Creation: ■ myArray = new int[4][3]; // OR ■ int myArray [][] = new int[4][3];■ Initialisation: ■ Single Value; ■ myArray[0][0] = 10; ■ Multiple values: ■ int tableA[2][3] = {{10, 15, 30}, {14, 30, 33}}; ■ int tableA[][] = {{10, 15, 30}, {14, 30, 33}}; 173

174. Variable Size Arrays■ Java treats multidimensional arrays as “arrays of arrays”. It is possible to declare a 2D arrays as follows: ■ int a[][] = new int [3][]; ■ a[0]= new int [3]; ■ a[1]= new int [2]; ■ a[2]= new int [4]; 174

175. Try: Write a program to Add to Matrix ■ Define 2 dimensional matrix variables: ■ Say: int a[][], b[][]; ■ Define their size to be 2x3 ■ Initialise like some values ■ Create a matrix c to storage sum

value ■ c[0][0] = a[0][0] + b[0][0] ■ Print the contents of result matrix. 175

176. Arrays of Objects■ Arrays can be used to store objects Circle[] circleArray; circleArray = new Circle[25];■ The above statement creates an array that can store references to 25 Circle objects.■ Circle objects are not created. 176

177. Arrays of Objects■ Create the Circle objects and stores them in the array. ■ //declare an array for Circle Circle circleArray[] = new Circle[25]; int r = 0; // create circle objects and store in array for (r=0; r <25; r++) circleArray[r] = new Circle(r); 177

178. String Operations in Java 178

179. Introduction■ String manipulation is the most common operation performed in Java programs. The easiest way to represent a String (a sequence of characters) is by using an array of characters. ■ Example: ■ char place[] = new char[4]; ■ place[0] = 'J'; ■ place[1] = 'a'; ■ place[2] = 'v'; ■ place[3] = 'a';■ Although character arrays have the advantage of being able to query their length, they themselves are too primitive and don't support a range of common string operations. For example, copying a string, searching for specific pattern etc.■ Recognising the importance and common usage of String manipulation in large software projects, Java supports String as one of the fundamental data type at the language level. Strings related book keeping operations (e.g., end of string) are handled automatically. 179

180. String Operations in Java■ Following are some useful classes that Java provides for String operations. ■ String Class ■ StringBuffer Class ■ StringTokenizer Class 180

181. String Class■ String class provides many operations for manipulating strings. ■ Constructors ■ Utility ■ Comparisons ■ Conversions■ String objects are read-only (immutable) 181

182. Strings Basics■ Declaration and Creation: ■ String stringName; ■ stringName = new String ("string value"); ■ Example: ■ String city; ■ city = new String ("Bangalore"); ■ Length of string can be accessed by invoking length() method defined in String class: ■ int len = city.length(); 182

183. String operations and Arrays■ Java Strings can be concatenated using the + operator. ■ String city = "New" + "York"; ■ String city1 = "Delhi"; ■ String city2 = "New "+city1;■ Strings Arrays ■ String city[] = new String[5]; ■ city[0] = new String("Melbourne"); ■ city[1] = new String("Sydney"); ■ … ■ String megacities[] = {"Brisbane", "Sydney", "Melbourne", "Adelaide", "Perth"}; 183

184. String class - Constructorspublic String() Constructs an empty String.Public String(String value) Constructs a new string copying the specified string. 184

185. String – Some useful operationspublic int length() Returns the length of the string.public charAt(int index) Returns the character at the specified location (index)public int compareTo(String Compare the Strings.anotherString)public intcompareToIgnoreCase(StringanotherString) 185

186. String – Some useful operationspublic String replace(char Returns a new string with

alloldChar, char newChar) instances of the oldChar replaced with newChar.public trim() Trims leading and trailing white spaces.public String toLowerCase() Changes as specified.public String toUpperCase() 186

187. String Class - example// StringDemo.java: some operations on stringsclass StringDemo { public static void main(String[] args) { String s = new String("Have a nice Day"); // String Length = 15 System.out.println("String Length = " + s.length()); // Modified String = Have a Good Day System.out.println("Modified String = " + s.replace(n, N)); // Converted to Uppercse = HAVE A NICE DAY" System.out.println("Converted to Uppercase = " + s.toUpperCase()); // Converted to Lowercase = have a nice day" System.out.println("Converted to Lowercase = " + s.toLowerCase()); }} 187

188. StringDemo Output■ [raj@mundroo] Arrays [1:130] java StringDemoString Length = 15Modified String = Have a Nice DayConverted to Uppercase = HAVE A NICE DAYConverted to Lowercase = have a nice day■ [raj@mundroo] Arrays [1:131] 188

189. Summary■ Arrays allows grouping of sequence of related items.■ Java supports powerful features for declaring, creating, and manipulating arrays in efficient ways.■ Each items of arrays of arrays can have same or variable size.■ Java provides enhanced support for manipulating strings and manipulating them appears similar to manipulating standard data type variables. 189

190. Final and Abstract Classes 190

191. Restricting Inheritance Parent Inherited capability Child 191

192. Final Members: A way for PreventingOverriding of Members in Subclasses ■ All methods and variables can be overridden by default in subclasses. ■ This can be prevented by declaring them as final using the keyword “final” as a modifier. For example: ■ final int marks = 100; ■ final void display(); ■ This ensures that functionality defined in this method cannot be altered any. Similarly, the value of a final variable cannot be altered. 192

193. Final Classes: A way for Preventing Classes being extended ■ We can prevent an inheritance of classes by other classes by declaring them as final classes. ■ This is achieved in Java by using the keyword final as follows: final class Marks { // members } final class Student extends Person { // members } ■ Any attempt to inherit these classes will cause an error. 193

194. Abstract Classes■ When we define a class to be “final”, it cannot be extended. In certain situation, we want to properties of classes to be always extended and used. Such classes are called Abstract Classes.■ An Abstract class is a conceptual class.■ An Abstract class cannot be instantiated – objects cannot be created.■ Abstract classes provides a common root for a group of classes, nicely tied together in a package: 194

195. Abstract Class Syntaxabstract class ClassName{ ... … abstract Type MethodName1(); … … Type Method2() { // method body }}■ When a class contains one or more abstract methods, it should be declared as abstract class.■ The abstract methods of an abstract class must be defined in its subclass.■ We cannot declare abstract constructors or abstract static methods. 195

196. Abstract Class -Example■ Shape is a abstract class. Shape Circle Rectangle 196

197. The Shape Abstract Class public abstract class Shape { public abstract double area(); public void move() { // non-abstract method // implementation } }■ Is the following statement valid? ■ Shape s = new Shape();■ No. It is illegal because the Shape class is an abstract class, which cannot be instantiated to create its objects. 197

198. Abstract Classespublic Circle extends Shape { protected double r; protected static final double PI =3.1415926535; public Circle() { r = 1.0;) public double area() { return PI * r * r; }…}public Rectangle extends Shape { protected double w, h; public Rectangle() { w = 0.0; h=0.0; } public double area() { return w * h; }} 198

199. Abstract Classes Properties■ A class with one or more abstract methods is automatically abstract and it cannot be instantiated.■ A class declared abstract, even with no abstract methods can not be instantiated.■ A subclass of an abstract class can be instantiated if it overrides all abstract methods by implementation them.■ A subclass that does not implement all of the superclass abstract methods is itself abstract; and it cannot be instantiated. 199

200. Summary■ If you do not want (properties of) your class to be extended or inherited by other classes, define it as a final class. ■ Java supports this is through the keyword “final”. ■ This is applied to classes.■ You can also apply the final to only methods if you do not want anyone to override them.■ If you want your class (properties/methods) to be extended by all those who want to use, then define it as an abstract class or define one or more of its methods as abstract methods. ■ Java supports this is through the keyword “abstract”. ■ This is applied to methods only. ■ Subclasses should implement abstract methods; otherwise, they cannot be instantiated. 200

201. InterfacesDesign Abstraction and a way for loosing realizing Multiple Inheritance 201

202. Interfaces■ Interface is a conceptual entity similar to a Abstract class.■ Can contain only constants (final variables) and abstract method (no implementation) - Different from Abstract classes.■ Use when a number of classes share a common interface.■ Each class should implement the interface. 202

203. Interfaces: An informal way of realising multiple inheritance■ An interface is basically a kind of class—it contains methods and variables, but they have to be only abstract classes and final fields/variables.■ Therefore, it is the responsibility of the class that implements an interface to supply the code for methods.■ A class can implement any number of interfaces, but cannot extend more than one class at a time.■ Therefore, interfaces are considered as an informal way of realising multiple inheritance in Java. 203

204. Interface - Example <<Interface>> Speaker speak()Politician Priest Lecturerspeak() speak() speak() 204

205. Interfaces Definition■ Syntax (appears like abstract class): interface InterfaceName { // Constant/Final Variable Declaration // Methods Declaration – only method body }■ Example: interface

Speaker { public void speak( ); } 205

206. Implementing Interfaces■ Interfaces are used like super-classes who properties are inherited by classes. This is achieved by creating a class that implements the given interface as follows:class ClassName implements InterfaceName [, InterfaceName2, …]{ // Body of Class} 206

207. Implementing Interfaces Example class Politician implements Speaker { public void speak(){ System.out.println(“Talk politics”); } } class Priest implements Speaker { public void speak(){ System.out.println(“Religious Talks”); } } class Lecturer implements Speaker { public void speak(){ System.out.println(“Talks Object Oriented Design and Programming!”); } } 207

208. Extending Interfaces■ Like classes, interfaces can also be extended. The new sub-interface will inherit all the members of the superinterface in the manner similar to classes. This is achieved by using the keyword extends as follows: interface InterfaceName2 extends InterfaceName1 { // Body of InterfaceName2 } 208

209. Inheritance and Interface Implementation■ A general form of interface implementation: class ClassName extends SuperClass implements InterfaceName [, InterfaceName2, …] { // Body of Class }■ This shows a class can extended another class while implementing one or more interfaces. It appears like a multiple inheritance (if we consider interfaces as special kind of classes with certain restrictions or special features). 209

210. Student Assessment Example■ Consider a university where students who participate in the national games or Olympics are given some grace marks. Therefore, the final marks awarded = Exam_Marks + Sports_Grace_Marks. A class diagram representing this scenario is as follow: Student Sports extends Exam implements extends Results 210

211. Software Implementationclass Student{ // student no and access methods}interface Sport{ // sports grace marks (say 5 marks) and abstract methods}class Exam extends Student{ // example marks (test1 and test 2 marks) and access methods}class Results extends Exam implements Sport{ // implementation of abstract methods of Sport interface // other methods to compute total marks = test1+test2+sports_grace_marks; // other display or final results access methods} 211

212. Summary■ Data and methods may be hidden or encapsulated within a class by specifying the private or protected visibility modifiers.■ An abstract method has no method body. An abstract class contains abstract methods.■ An interface is a collection of abstract methods and constants. A class implements an interface by declaring it in its implements clause, and providing a method body for each abstract method. 212

213. Exceptions:Way of Handling Errors 213

214. Introduction■ Rarely does a program runs successfully at its very first attempt.■ It is common to make mistakes while developing as well as typing a program.■ Such mistakes are categorised as: ■ syntax errors - compilation errors. ■ semantic errors– leads to programs producing unexpected outputs. ■ runtime errors – most often lead to abnormal termination of programs or even cause the system to

crash. 214

215. Common Runtime Errors■ Dividing a number by zero.■ Accessing an element that is out of bounds of an array.■ Trying to store incompatible data elements.■ Using negative value as array size.■ Trying to convert from string data to a specific data value (e.g., converting string “abc” to integer value).■ File errors: ■ opening a file in “read mode” that does not exist or no read permission ■ Opening a file in “write/update mode” which has “read only” permission.■ Corrupting memory: - common with pointers■ Any more …. 215

216. Without Error Handling – Example 1 class NoErrorHandling{ public static void main(String[] args){ int a,b; a = 7; b = 0;Program does not reach here System.out.println(“Result is “ + a/b); System.out.println(“Program reached this line”); } } No compilation errors. While running it reports an error and stops without executing further statements: java.lang.ArithmeticException: / by zero at Error2.main(Error2.java:10) 216

217. Traditional way of Error Handling - Example 2 class WithErrorHandling{ public static void main(String[] args){ int a,b; a = 7; b = 0; if (b != 0){ System.out.println(“Result is “ + a/b); } else{Program reaches here System.out.println(“ B is zero); } System.out.println(“Program is complete”); } } 217

218. Error Handling■ Any program can find itself in unusual circumstances – Error Conditions.■ A “good” program should be able to handle these conditions gracefully.■ Java provides a mechanism to handle these error condition - exceptions 218

219. Exceptions■ An exception is a condition that is caused by a runtime error in the program.■ Provide a mechanism to signal errors directly without using flags.■ Allow errors to be handled in one central part of the code without cluttering code. 219

220. Exceptions and their Handling■ When the JVM encounters an error such as divide by zero, it creates an exception object and throws it – as a notification that an error has occurred.■ If the exception object is not caught and handled properly, the interpreter will display an error and terminate the program.■ If we want the program to continue with execution of the remaining code, then we should try to catch the exception object thrown by the error condition and then take appropriate corrective actions. This task is known as exception handling. 220

221. Common Java Exceptions■ ArithmeticException■ ArrayIndexOutOfBoundException■ ArrayStoreException■ FileNotFoundException■ IOException – general I/O failure■ NullPointerException – referencing a null object■ OutOfMemoryException■ SecurityException – when applet tries to perform an action not allowed by the browser’s security setting.■ StackOverflowException■ StringIndexOutOfBoundException 221

222. Exceptions in Java■ A method can signal an error condition by throwing an exception – throws■ The calling method can transfer control to a exception handler by catching an exception - try, catch■ Clean up can be done by - finally 222

223. Exception Handling Mechanism try Block Statements that causes an exception Throws

exception Object catch Block Statements that handle the exception 223

224. Syntax of Exception Handling Code … … try { // statements } catch(Exception-Type e) { // statements to process exception } 224

225. With Exception Handling - Example 3 class WithExceptionHandling{ public static void main(String[] args){ int a,b; float r; a = 7; b = 0; try{ r = a/b; System.out.println(“Result is “ + r);Program Reaches here} catch(ArithmeticException e){ System.out.println(“ B is zero); } System.out.println(“Program reached this line”); } } 225

226. Finding a Sum of Integer Values Passed as Command Line Parameters// ComLineSum.java: adding command line parametersclass ComLineSum{ public static void main(String args[]) { int InvalidCount = 0; int number, sum = 0; for(int i = 0; i < args.length; i++) { try { number = Integer.parseInt(args[i]); } catch(NumberFormatException e) { InvalidCount++; System.out.println("Invalid Number: "+args[i]); continue;//skip the remaining part of loop } sum += number; } System.out.println("Number of Invalid Arguments = "+InvalidCount); System.out.println("Number of Valid Arguments = "+(args.length-InvalidCount)); System.out.println("Sum of Valid Arguments = "+sum); }} 226

227. Sample Runs[raj@mundroo] java ComLineSum 1 2Number of Invalid Arguments = 0Number of Valid Arguments = 2Sum of Valid Arguments = 3[raj@mundroo] java ComLineSum 1 2 abcInvalid Number: abcNumber of Invalid Arguments = 1Number of Valid Arguments = 2Sum of Valid Arguments = 3 227

228. Multiple Catch Statements■ If a try block is likely to raise more than one type of exceptions, then multiple catch blocks can be defined as follows: … … try { // statements } catch(Exception-Type1 e) { // statements to process exception 1 } .. .. catch(Exception-TypeN e) { // statements to process exception N } … 228

229. finally block■ Java supports definition of another block called finally that be used to handle any exception that is not caught by any of the previous statements. It may be added immediately after the try block or after the last catch block: … try { // statements } catch(Exception-Type1 e) { // statements to process exception 1 } .. .. finally { …. }■ When a finally is defined, it is executed regardless of whether or not an exception is thrown. Therefore, it is also used to perform certain house keeping operations such as closing files and releasing system resources. 229

230. With Exception Handling - Example 4 class WithExceptionCatchThrow{ public static void main(String[] args){ int a,b; float r; a = 7; b = 0; try{ r = a/b; System.out.println(“Result is “ + r); } Program Does Not reach here catch(ArithmeticException e){when exception occurs System.out.println(“ B is zero); throw e; } System.out.println(“Program is complete”); } } 230

231. With Exception Handling - Example 5 class WithExceptionCatchThrowFinally{ public static void main(String[] args){ int a,b; float r; a = 7; b = 0; try{ r = a/b; System.out.println(“Result is “ + r); }Program reaches here catch(ArithmeticException e){ System.out.println(“ B is zero); throw e; } finally{

System.out.println(“Program is complete”); } } } 231

232. Summary■ A good programs does not produce unexpected results.■ It is always a good practice to check for potential problem spots in programs and guard against program failures.■ Exceptions are mainly used to deal with runtime errors.■ Exceptions also aid in debugging programs.■ Exception handling mechanisms can effectively used to locate the type and place of errors. 232

233. Summary■ Try block, code that could have exceptions / errors■ Catch block(s), specify code to handle various types of exceptions. First block to have appropriate type of exception is invoked.■ If no ‘local’ catch found, exception propagates up the method call stack, all the way to main()■ Any execution of try, normal completion, or catch then transfers control on to finally block 233

234. Streams and Input/Output Files 234

235. Introduction■ So far we have used variables and arrays for storing data inside the programs. This approach poses the following limitations: ■ The data is lost when variable goes out of scope or when the program terminates. That is data is stored in temporary/mail memory is released when program terminates. ■ It is difficult to handle large volumes of data.■ We can overcome this problem by storing data on secondary storage devices such as floppy or hard disks.■ The data is stored in these devices using the concept of Files and such data is often called persistent data. 235

236. File Processing■ Storing and manipulating data using files is known as file processing.■ Reading/Writing of data in a file can be performed at the level of bytes, characters, or fields depending on application requirements.■ Java also provides capabilities to read and write class objects directly. The process of reading and writing objects is called object serialisation. 236

237. C Input/Output RevisionFILE* fp;fp = fopen(“In.file”, “rw”);fscanf(fp, ……);frpintf(fp, …..);fread(……., fp);fwrite(……., fp); 237

238. I/O and Data Movement■ The flow of data into a program (input) may come from different devices such as keyboard, mouse, memory, disk, network, or another program.■ The flow of data out of a program (output) may go to the screen, printer, memory, disk, network, another program.■ Both input and output share a certain common property such as unidirectional movement of data – a sequence of bytes and characters and support to the sequential access to the data. 238

239. Streams■ Java Uses the concept of Streams to represent the ordered sequence of data, a common characteristic shared by all I/O devices.■ Streams presents a uniform, easy to use, object oriented interface between the program and I/O devices.■ A stream in Java is a path along which data flows (like a river or pipe along which water flows). 239

240. Stream Types■ The concepts of sending data from one stream to Input Stream another (like a pipe reads feeding into another Source Program pipe) has made streams powerful tool for file processing. Output Stream■ Connecting streams can Program Source also act as filters. writes■ Streams are classified into two basic types: ■ Input Steam ■ Output Stream 240

241. Java Stream Classes■ Input/Output related classes are defined in java.io package.■ Input/Output in Java is defined in terms of streams.■ A stream is a sequence of data, of no particular length.■ Java classes can be categorised into two groups based on the data type one which they operate: ■ Byte streams ■ Character Streams 241

242. Streams Byte Streams Character streamsOperated on 8 bit (1 Operates on 16-bitbyte) data. (2 byte) unicode characters.Input streams/Output Readers/ Writersstreams 242

243. Classification of Java Stream Classes Byte Stream Character Stream classes classes 243

244. Byte Input Streams InputStream ObjectInputStream SequenceInputStream ByteArrayInputStream PipedInputStream FilterInputStream PushbackInputStream BufferedInputStreamDataInputStream 244

245. Byte Input Streams - operationspublic abstract int read() Reads a byte and returns as a integer 0-255public int read(byte[] buf, int Reads and stores the bytesoffset, int count) in buf starting at offset. Count is the maximum read.public int read(byte[] buf) Same as previous offset=0 and length=buf.length() 245

246. Byte Input Stream - example■ Count total number of bytes in the file import java.io.*; class CountBytes { public static void main(String[] args) throws FileNotFoundException, IOException { FileInputStream in; in = new FileInputStream(“InFile.txt”); int total = 0; while (in.read() != -1) total++; System.out.println(total + “ bytes”); } } 246

247. What happens if the file did not exist ■ JVM throws exception and terminates the program since there is no exception handler defined. [raj@mundroo] Streams [1:165] java CountBytes Exception in thread "main" java.io.FileNotFoundException: FileIn.txt (No such file or directory) at java.io.FileInputStream.open(Native Method) at java.io.FileInputStream.<init>(FileInputStream.java:64) at CountBytes.main(CountBytes.java:12) 247

248. Byte Output Streams OutputStream ObjectOutputStream SequenceOutputStream ByteArrayOutputStream PipedOutputStream FilterOutputStream PrintStream BufferedOutputStreamDataOutputStream 248

249. Byte Output Streams - operations public abstract void write(int Write b as bytes. b) public void write(byte[] buf, Write count bytes starting int offset, int count) from offset in buf. public void write(byte[] buf) Same as previous offset=0 and count = buf.length() 249

250. Byte Output Stream - example■ Read from standard in and write to standard out import java.io.*; class ReadWrite { public static void main(string[] args) throws IOException { int b; while ((b = System.in.read()) != -1) { System.out.write(b); } } 250

251. Summary■ Streams provide uniform interface for managing I/O operations in Java irrespective of device types.■ Java supports classes for handling Input Steams and Output steams via java.io package.■ Exceptions supports handling of errors and their propagation during file operations.

251

252. Graphical User Interface (GUI) ApplicationsAbstract Windowing Toolkit (AWT) Events Handling Applets 252

253. Introduction■ Java began as a language to be integrated with browsers.■ But it as evolved as a powerful language for developing stand-alone graphical applications and also server-side applications.■ Today, Java has large and powerful libraries to deal with 2D and 3D graphics and imaging, as well as the ability to build complex client-side interactive systems.■ Our focus: Simple GUI apps and Applets and Graphics. More on graphics in your 3rd year subject on “Interactive Computing”. 253

254. AWT - Abstract Windowing Toolkit■ Single Windowing Interface on Multiple Platforms■ Supports functions common to all window systems■ Uses Underlying Native Window system■ AWT provides ■ GUI widgets ■ Event Handling ■ Containers for widgets ■ Layout managers ■ Graphic operations 254

255. AWT - Abstract Window Toolkit■ Portable GUI - preserves native look and feel■ Standard GUI Components (buttons…)■ Containers - Panels, Frames, Dialogs■ Graphics class for custom drawing■ Layouts responsible for actual positioning of components: ■ BorderLayout, GridLayout, FlowLayout, Null layout 255

256. Adding Components via Layouts 256

257. Building Graphical User Interfaces ■ import java.awt.*; ■ Assemble the GUI ■ use GUI components, ■ basic components (e.g., Button, TextField) ■ containers (Frame, Panel) ■ set the positioning of the components ■ use Layout Managers ■ Attach events 257

258. A sample GUI programimport java.awt.*;public class MyGui{ public static void main(String args[]) { Frame f = new Frame ("My Frame"); Button b = new Button("OK"); TextField tf = new TextField("Programming in Java", 20); f.setLayout(new FlowLayout()); f.add(b); f.add(tf); f.setSize(300, 300); f.setVisible(true); }} 258

259. Output 259

260. Events■ Each GUI component (e.g., a Button) that wishes to respond to an event type (e.g., click), must register an event handler, called a Listener.■ The listener is an object of a "Listener" interface.■ A Listener class can be created by subclassing (through "implements") one of Listener interfaces (all listener inrefaces are in the java.awt.event package = > must import java.awt.event.*;)■ The registration of the listener is done by a call to a method such as addActionListener(<Listener Object>). Each GUI component class has one or more such add…() methods, where applicable. 260

261. Events b.addActionListener( );Button method to add a listener listener objectFrame f.addWindowListener( ); 261

262. Listener Interfaces in java.awt.event.* ■ [1] ActionListener ■ [2] ItemListener ■ [3]

MouseMotionListener ■ [4] MouseListener ■ [5] KeyListener ■ [6] FocusListener ■ [7] AdjustmentListener ■ [8] ComponentListener ■ [9] WindowListener ■ [10] ContainerListener ■ [11] TextListener 262

263. Listener Interfaces■ Each listener interface has methods that need to be implemented for handling different kinds of events.■ For example 1, the ActionListener interface has a method actionPerformed() button component is operated.■ For example2, the MouseMotionListener interface has two methods: ■ 1) mouseDragged(MouseEvent) - Invoked when a mouse button is pressed on a component and then dragged. ■ 2) mouseMoved(MouseEvent) - Invoked when the mouse button has been moved on a component (with no buttons down). 263

264. Implementing the ActionListener Interface and attaching an event handler to a button import java.awt.*; import java.awt.event.*; public class MyGui1 { public static void main(String args[]) { Frame f = new Frame ("My Frame"); MyGuiAction ga = new MyGuiAction(f); } } class MyGuiAction implements ActionListener { static int count = 0; Button b; TextField tf; MyGuiAction(Frame f) { b = new Button("OK"); b.addActionListener(this); tf = new TextField("Hello Java", 20); f.setLayout(new FlowLayout()); f.add(b); f.add(tf); f.setSize(300, 300); f.setVisible(true); } public void actionPerformed(ActionEvent e) { if(e.getSource() == b) { count++; System.out.println("Button is Pressed"); tf.setText("Hello Java Click "+count); } } 264 }

265. Output and Clicks on “OK” Button Exec started 1st click on OK button 2nd click on OK button 265

266. BorderLayout Exampleimport java.awt.*;public class MyGui2{ public static void main(String args[]) { Frame f = new Frame ("My Frame"); f.setLayout(new BorderLayout()); // Add text field to top f.add("North",new TextField()); // Create the panel with buttons at the bottom... Panel p = new Panel(); // FlowLayout p.add(new Button("OK")); p.add(new Button("Cancel")); f.add("South",p); f.add("Center", new TextField("Center region")); f.setSize(300, 300); f.setVisible(true); }} 266

267. Output 267

268. Applets ProgrammingEnabling Application Delivery Via the Web 268

269. Introduction■ Applets are small Java programs that are embedded in Web pages.■ They can be transported over the Internet from one computer (web server) to another (client computers).■ They transform web into rich media and support the delivery of applications via the Internet. 269

270. Applet: Making Web Interactive and Application Delivery Media 1 2 3 4 5 APPLET hello.class Create Accessing The browser Development AT SUN’S Applet from creates “hello.java” WEB tag in Your Organisation a new AT SERVER HTML window and SUN.COM document a new thread and then runs the code Hello Java <app= “Hello”> The Internet Hello 270

271. How Applets Differ from Applications ■ Although both the Applets and stand-alone applications are Java programs, there are certain restrictions are imposed on Applets due to security concerns: ■ Applets don’t use the main() method, but when they are load, automatically call certain

methods (init, start, paint, stop, destroy). ■ They are embedded inside a web page and executed in browsers. ■ They cannot read from or write to the files on local computer. ■ They cannot communicate with other servers on the network. ■ They cannot run any programs from the local computer. ■ They are restricted from using libraries from other languages. ■ The above restrictions ensures that an Applet cannot do any damage to the local system. 271

272. Building Applet Code: An Example //HelloWorldApplet.java import java.applet.Applet; import java.awt.*; public class HelloWorldApplet extends Applet { public void paint(Graphics g) { g.drawString ("Hello World of Java!",25, 25); } } 272

273. Embedding Applet in Web Page <HTML> <HEAD> <TITLE> Hello World Applet </TITLE> </HEAD> <body> <h1>Hi, This is My First Java Applet on the Web!</h1> <APPLET CODE="HelloWorldApplet.class" width=500 height=400> </APPLET> </body> </HTML> 273

274. Accessing Web page (runs Applet) 274

275. Applet Life Cycle■ Every applet inherits a set of default behaviours from the Applet class. As a result, when an applet is loaded, it undergoes a series of changes in its state. The applet states include: ■ Initialisation – invokes init() ■ Running – invokes start() ■ Display – invokes paint() ■ Idle – invokes stop() ■ Dead/Destroyed State – invokes destroy() 275

276. Applet States■ Initialisation – invokes init() – only once ■ Invoked when applet is first loaded.■ Running – invokes start() – more than once ■ For the first time, it is called automatically by the system after init() method execution. ■ It is also invoked when applet moves from idle/stop() state to active state. For example, when we return back to the Web page after temporary visiting other pages.■ Display – invokes paint() - more than once ■ It happens immediately after the applet enters into the running state. It is responsible for displaying output.■ Idle – invokes stop() - more than once ■ It is invoked when the applet is stopped from running. For example, it occurs when we leave a web page.■ Dead/Destroyed State – invokes destroy() - only once ■ This occurs automatically by invoking destroy() method when we quite the browser. 276

277. Applet Life Cycle Diagram init()Begin Born stop() start() Running Idle destroy() paint() start() Dead End 277

278. Passing Parameters to Applet<HTML><HEAD><TITLE> Hello World Applet</TITLE></HEAD><body><h1>Hi, This is My First Communicating Applet on the Web!</h1><APPLET CODE="HelloAppletMsg.class" width=500 height=400> <PARAM NAME="Greetings" VALUE="Hello Friend, How are you?"></APPLET></body></HTML> 278

279. Applet Program Accepting Parameters//HelloAppletMsg.javaimport java.applet.Applet;import java.awt.*;public class HelloAppletMsg extends Applet { String msg; public void init() { msg = getParameter("Greetings"); if(msg == null) msg = "Hello"; } public void paint(Graphics g) { g.drawString (msg,10, 100); }} This is name of parameter specified in PARAM tag; This method returns the value of paramter. 279

280. HelloAppletMsg.html 280

281. What happen if we don’t pass parameter? See HelloAppletMsg1.html<HTML><HEAD><TITLE> Hello World Applet</TITLE></HEAD><body><h1>Hi, This is My First Communicating Applet on the Web!</h1><APPLET CODE="HelloAppletMsg.class" width=500 height=400></APPLET></body></HTML> 281

282. getParameter() returns null. Some default value may be used. 282

283. Displaying Numeric Values//SumNums.javaimport java.applet.Applet;import java.awt.*;public class SumNums extends Applet { public void paint(Graphics g) { int num1 = 10; int num2 = 20; int sum = num1 + num2; String str = "Sum: "+String.valueOf(sum); g.drawString (str,100, 125); }} 283

284. SunNums.html<HTML><HEAD><TITLE> Hello World Applet</TITLE></HEAD><body><h1>Sum of Numbers</h1><APPLET CODE="SumNums.class" width=500 height=400></APPLET></body></HTML> 284

285. Applet – Sum Numbers 285

286. Interactive Applets■ Applets work in a graphical environment. Therefore, applets treats inputs as text strings.■ We need to create an area on the screen in which use can type and edit input items.■ We can do this using TextField class of the applet package.■ When data is entered, an event is generated. This can be used to refresh the applet output based on input values. 286

287. Interactive Applet Program..(cont) //SumNumsInteractive..java import java.applet.Applet; import java.awt.*; public class SumNumsInteractive extends Applet public void paint(Graphics g) { { TextField text1, text2; int num1 = 0; public void init() int num2 = 0; { int sum; text1 = new TextField(10); String s1, s2, s3; text2 = new TextField(10); text1.setText("0"); g.drawString("Input a number in each box ", 10, 50); text2.setText("0"); try add(text1); { add(text2); s1 = } text1.getText(); public boolean action(Event ev, Object obj) num1 = { Integer.parseInt(s1); repaint(); s2 = text2.getText(); return true; num2 = } Integer.parseInt(s2); } catch(Exception e1) ➔➔➔ next code is here ➔➔➔ {} sum = num1 + num2; String str = "THE SUM IS: "+String.valueOf(sum); g.drawString (str,100, 125); } } 287

288. Interactive Applet Execution 288

289. Summary■ Applets are designed to operate in Internet and Web environment. They enable the delivery of applications via the Web.■ This is demonstrate by things that we learned in this lecture such as: ■ How do applets differ from applications? ■ Life cycles of applets ■ How to design applets? ■ How to execute applets? ■ How to provide interactive inputs? 289

290. Network Programming and Java Sockets 290

291. Agenda■ Introduction■ Elements of Client Server Computing■ Networking Basics■ Understanding Ports and Sockets■ Java Sockets ■ Implementing a Server ■ Implementing a Client■ Sample Examples■ Conclusions 291

292. Introduction■ Internet and WWW have emerged as global ubiquitous media for communication and changing the way we conduct science, engineering, and commerce.■ They also changing the way we learn, live, enjoy, communicate, interact, engage, etc. It appears like the modern life activities are getting completely centered around the Internet. 292

293. Internet Applications Serving Local and Remote Users PC client Internet Server Local Area Network PDA 293

294. Internet & Web as a delivery Vehicle 294

295. Increased demand for Internet applications■ To take advantage of opportunities presented by the Internet, businesses are continuously seeking new and innovative ways and means for offering their services via the Internet.■ This created a huge demand for software designers with skills to create new Internet- enabled applications or migrate existing/legacy applications on the Internet platform.■ Object-oriented Java technologies—Sockets, threads, RMI, clustering, Web services-- have emerged as leading solutions for creating portable, efficient, and maintainable large and complex Internet applications. 295

296. Elements of C-S Computing a client, a server, and network st e qu Client Re Server Re Network su lt Client machine Server machine 296

297. Networking Basics■ Applications Layer ■ TCP/IP Stack ■ Standard apps ■ HTTP ■ FTP ■ Telnet Application ■ User apps (http,ftp,telnet,…)■ Transport Layer Transport ■ TCP ■ UDP (TCP, UDP,..) ■ Programming Interface: Network ■ Sockets (IP,..)■ Network Layer ■ IP Link■ Link Layer (device driver,..) ■ Device drivers 297

298. Networking Basics■ TCP (Transport Control ■ TCP/IP Stack Protocol) is a connection-oriented Application protocol that provides a (http,ftp,telnet,…) reliable flow of data Transport between two computers. (TCP, UDP,..)■ Example applications: Network ■ HTTP (IP,..) ■ FTP Link ■ Telnet (device driver,..) 298

299. Networking Basics■ UDP (User Datagram ■ TCP/IP Stack Protocol) is a protocol that sends independent Application packets of data, called (http,ftp,telnet,…) datagrams, from one Transport computer to another with (TCP, UDP,..) no guarantees about Network arrival. (IP,..)■ Example applications: Link ■ Clock server (device driver,..) ■ Ping 299

300. Understanding Ports■ The TCP and UDP protocols use ports to P map incoming data to a o TCP particular process server r Client running on a computer. t app app app app port port port port TCP or UDP Packet Data port# data 300

301. Understanding Ports■ Port is represented by a positive (16-bit) integer value■ Some ports have been reserved to support common/well known services: ■ ftp 21/tcp ■ telnet 23/tcp ■ smtp 25/tcp ■ login 513/tcp■ User level process/services generally use port number value >= 1024 301

302. Sockets■ Sockets provide an interface for programming networks at the transport layer.■

Network communication using Sockets is very much similar to performing file I/O ■ In fact, socket handle is treated like file handle. ■ The streams used in file I/O operation are also applicable to socket-based I/O■ Socket-based communication is programming language independent. ■ That means, a socket program written in Java language can also communicate to a program written in Java or non-Java socket program. 302

303. Socket Communication■ A server (program) runs on a specific computer and has a socket that is bound to a specific port. The server waits and listens to the socket for a client to make a connection request. Connection request port server Client 303

304. Socket Communication■ If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bounds to a different port. It needs a new socket (consequently a different port number) so that it can continue to listen to the original socket for connection requests while serving the connected client. port server port Client port Connection 304

305. Sockets and Java Socket Classes■ A socket is an endpoint of a two-way communication link between two programs running on the network.■ A socket is bound to a port number so that the TCP layer can identify the application that data destined to be sent.■ Java's .net package provides two classes: ■ Socket – for implementing a client ■ ServerSocket – for implementing a server 305

306. Java Sockets Server ServerSocket(1234) Output/write stream Input/read stream Client Socket("128.250.25.158", 1234)It can be host_name like "mandroo.cs.mu.oz.au" 306

307. Implementing a Server1. Open the Server Socket: ServerSocket server; DataOutputStream os; DataInputStream is; server = new ServerSocket(PORT);2. Wait for the Client Request: Socket client = server.accept();3. Create I/O streams for communicating to the client is = new DataInputStream(client.getInputStream()); os = new DataOutputStream(client.getOutputStream());4. Perform communication with client Receive from client: String line = is.readLine(); Send to client: os.writeBytes("Hellon");5. Close sockets: client.close();For multithreaded server: while(true) { i. wait for client requests (step 2 above) ii. create a thread with "client" socket as parameter (the thread creates streams (as in step (3) and does communication as stated in (4). Remove thread once service is provided.} 307

308. Implementing a Client1. Create a Socket Object: client = new Socket(server, port_id);2. Create I/O streams for communicating with the server. is = new DataInputStream(client.getInputStream()); os = new DataOutputStream(client.getOutputStream());3. Perform I/O or communication with the server: ■ Receive data from the server: String line = is.readLine(); ■ Send data to the server: os.writeBytes("Hellon");4. Close the socket when done: client.close(); 308

309. A simple server (simplified code) // SimpleServer.java: a simple server program import java.net.*; import java.io.*; public class SimpleServer { public static void main(String args[]) throws IOException { // Register service on port 1234 ServerSocket s = new ServerSocket(1234); Socket s1=s.accept(); // Wait and accept a connection // Get a communication stream associated with the

socket OutputStream s1out = s1.getOutputStream(); DataOutputStream dos = new DataOutputStream (s1out); // Send a string! dos.writeUTF("Hi there"); // Close the connection, but not the server socket dos.close(); s1out.close(); s1.close(); } } 309

310. A simple client (simplified code) // SimpleClient.java: a simple client program import java.net.*; import java.io.*; public class SimpleClient { public static void main(String args[]) throws IOException { // Open your connection to a server, at port 1234 Socket s1 = new Socket("mundroo.cs.mu.oz.au",1234); // Get an input file handle from the socket and read the input InputStream s1In = s1.getInputStream(); DataInputStream dis = new DataInputStream(s1In); String st = new String (dis.readUTF()); System.out.println(st); // When done, just close the connection and exit dis.close(); s1In.close(); s1.close(); } } 310

311. Run■ Run Server on mundroo.cs.mu.oz.au ■ [raj@mundroo] java SimpleServer &■ Run Client on any machine (including mundroo): ■ [raj@mundroo] java SimpleClient Hi there■ If you run client when server is not up: ■ [raj@mundroo] sockets [1:147] java SimpleClient Exception in thread "main" java.net.ConnectException: Connection refused at java.net.PlainSocketImpl.socketConnect(Native Method) at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:320) at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:133) at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:120) at java.net.Socket.<init>(Socket.java:273) at java.net.Socket.<init>(Socket.java:100) at SimpleClient.main(SimpleClient.java:6) 311

312. Socket Exceptionstry { Socket client = new Socket(host, port); handleConnection(client);}catch(UnknownHostException uhe) { System.out.println("Unknown host: " + host); uhe.printStackTrace();}catch(IOException ioe) {System.out.println("IOException: " + ioe); ioe.printStackTrace();} 312

313. ServerSocket & Exceptions■ public ServerSocket(int port) throws IOException ■ Creates a server socket on a specified port. ■ A port of 0 creates a socket on any free port. You can use getLocalPort() to identify the (assigned) port on which this socket is listening. ■ The maximum queue length for incoming connection indications (a request to connect) is set to 50. If a connection indication arrives when the queue is full, the connection is refused.■ Throws: ■ IOException - if an I/O error occurs when opening the socket. ■ SecurityException - if a security manager exists and its checkListen method doesnt allow the operation. 313

314. Server in Loop: Always up// SimpleServerLoop.java: a simple server program that runs forever in a single theadimport java.net.*;import java.io.*;public class SimpleServerLoop { public static void main(String args[]) throws IOException { // Register service on port 1234 ServerSocket s = new ServerSocket(1234); while(true) { Socket s1=s.accept(); // Wait and accept a connection // Get a communication stream associated with the socket OutputStream s1out = s1.getOutputStream(); DataOutputStream dos = new DataOutputStream (s1out); // Send a string! dos.writeUTF("Hi there"); // Close the connection, but not the server socket dos.close(); s1out.close(); s1.close(); } }} 314

315. Multithreaded Server: For Serving Multiple Clients Concurrently Client 1 Process Server Process Server Threads Internet Client 2 Process 315

316. Conclusion■ Programming client/server applications in Java is fun and challenging.■ Programming socket programming in Java is much easier than doing it in other languages such as C.■ Keywords: ■ Clients, servers, TCP/IP, port number, sockets, Java sockets 316

317. Multithreaded Programming in Java 317

318. Agenda■ Introduction■ Thread Applications■ Defining Threads■ Java Threads and States■ Examples 318

319. A single threaded programclass ABC{…. public void main(..) begin { … body .. end }} 319

320. A Multithreaded Program Main Thread start start startThread A Thread B Thread C Threads may switch or exchange data/results 320

321. Web/Internet Applications:Serving Many Users Simultaneously PC client Internet Server Local Area Network PDA 321

322. Multithreaded Server: For Serving Multiple Clients Concurrently Client 1 Process Server Process Server Threads ■ Internet Client 2 Process 322

323. Modern Applications need Threads (ex1):Editing and Printing documents in background. Printing Thread Printing Thread Editing Thread Editing Thread 323

324. Multithreaded/Parallel File Copy reader() reader() {{ writer() writer() -- -- -- -- -- -- -- -- -- buff[0] {{ -- buff[0] -- -- -- -- -- -- -- -- -- -- lock(buff[i]); lock(buff[i]); lock(buff[i]); buff[1] lock(buff[i]); read(src,buff[i]); read(src,buff[i]); buff[1] write(src,buff[i]); write(src,buff[i]); unlock(buff[i]); unlock(buff[i]); unlock(buff[i]); unlock(buff[i]); -- }} }} Cooperative Parallel Synchronized Cooperative Parallel Synchronized Threads Threads 324

325. Levels of Parallelism Code-Granularity Code-Granularity Code Item Code ItemSockets/ Task i-l Task i-l Task ii Task Task i+1 Task i+1 Large grain Large grain (task level)PVM/MPI (task level) Program Program func1 (()) func2 (()) func3 (()) func1 func2 func3 Medium grain Medium grain {{ {{ {{ (control level) (control level)Threads .... Function (thread) Function (thread) .... }} }} }} Fine grain Fine grain (data level) (data level) aa((00))=.. aa((11)=.. aa((22)=.. Loop (Compiler) Loop (Compiler)Compilers =.. bb((00))=.. =..)=.. bb((11)=..)=..)=.. bb((22)=..)=.. Very fine grain Very fine grain (multiple issue) (multiple issue) With hardware With hardwareCPU + + xx Load Load 325

326. Single and Multithreaded Processes threads are light-weight processes within a processSingle-threaded Process Multiplethreaded Threads of Process ExecutionSingle instruction stream Common Multiple instruction stream Address Space 326

327. Multithreading - Multiprocessors Process Parallelism Process Parallelism CPU P1 P1 P2 CPU P2 P3 CPU P3 time time No of execution process more the number of CPUs No of execution process more the number of CPUs 327

328. Multithreading on Uni-processor ■ Concurrency Vs Parallelism K Process Concurrency K

Process Concurrency P1 P1 P2 P2 CPU P3 P3 time time Number of Simultaneous execution units > number of Number of Simultaneous execution units > number of CPUs CPUs 328

329. What are Threads?■ A piece of code that run in concurrent with other threads.■ Each thread is a statically ordered sequence of instructions.■ Threads are being extensively used express concurrency on both single and multiprocessors machines.■ Programming a task having multiple threads of control – Multithreading or Multithreaded Programming. 329

330. Java Threads■ Java has built in thread support for Multithreading■ Synchronization■ Thread Scheduling■ Inter-Thread Communication: ■ currentThread start setPriority ■ yield run getPriority ■ sleep stop suspend ■ resume■ Java Garbage Collector is a low-priority thread 330

331. Threading Mechanisms...■ Create a class that extends the Thread class■ Create a class that implements the Runnable interface 331

332. 1st method: Extending Thread class■ Threads are implemented as objects that contains a method called run() class MyThread extends Thread { public void run() { // thread body of execution } }■ Create a thread: MyThread thr1 = new MyThread();■ Start Execution of threads: thr1.start(); 332

333. An exampleclass MyThread extends Thread { // the thread public void run() { System.out.println(" this thread is running ... "); }} // end class MyThreadclass ThreadEx1 { // a program that utilizes the thread public static void main(String [] args) { MyThread t = new MyThread(); // due to extending the Thread class (above) // I can call start(), and this will call // run(). start() is a method in class Thread. t.start(); } // end main()} // end class ThreadEx1 333

334. 2nd method: Threads by implementing Runnable interfaceclass MyThread implements Runnable{ ..... public void run() { // thread body of execution }}■ Creating Object: MyThread myObject = new MyThread();■ Creating Thread Object: Thread thr1 = new Thread(myObject);■ Start Execution: thr1.start(); 334

335. An exampleclass MyThread implements Runnable { public void run() { System.out.println(" this thread is running ... "); }} // end class MyThreadclass ThreadEx2 { public static void main(String [] args) { Thread t = new Thread(new MyThread()); // due to implementing the Runnable interface // I can call start(), and this will call run(). t.start(); } // end main()} // end class ThreadEx2 335

336. Life Cycle of Threadnew wait()start() sleep() suspend() blocked runnable non-runnable notify()stop() slept resume()dead unblocked 336

337. A Program with Three Java Threads ■ Write a program that creates 3 threads 337

338. Three threads example■ class A extends Thread■ {■ public void run()■ {■ for(int i=1;i<=5;i++)■ {■ System.out.println("t From ThreadA: i= "+i);■ }■ System.out.println("Exit from A");■ }■ }■ class B extends Thread■ {■ public void run()■ {■ for(int j=1;j<=5;j++)■ {■ System.out.println("t From ThreadB: j= "+j);■ }■ System.out.println("Exit from B");■ } 338■ }

339. ■ class C extends Thread■ {■ public void run()■ {■ for(int k=1;k<=5;k++)■ {■ System.out.println("t From ThreadC: k= "+k);■ }■ System.out.println("Exit from C");■ }■ }■ class ThreadTest■ {■ public static void main(String args[])■ {■ new A().start();■ new B().start();■ new C().start();■ }■ } 339

340. Run 1■ [raj@mundroo] threads [1:76] java ThreadTest From ThreadA: i= 1 From ThreadA: i= 2 From ThreadA: i= 3 From ThreadA: i= 4 From ThreadA: i= 5Exit from A From ThreadC: k= 1 From ThreadC: k= 2 From ThreadC: k= 3 From ThreadC: k= 4 From ThreadC: k= 5Exit from C From ThreadB: j= 1 From ThreadB: j= 2 From ThreadB: j= 3 From ThreadB: j= 4 From ThreadB: j= 5Exit from B 340

341. Run2■ [raj@mundroo] threads [1:77] java ThreadTest From ThreadA: i= 1 From ThreadA: i= 2 From ThreadA: i= 3 From ThreadA: i= 4 From ThreadA: i= 5 From ThreadC: k= 1 From ThreadC: k= 2 From ThreadC: k= 3 From ThreadC: k= 4 From ThreadC: k= 5Exit from C From ThreadB: j= 1 From ThreadB: j= 2 From ThreadB: j= 3 From ThreadB: j= 4 From ThreadB: j= 5Exit from BExit from A 341

342. Process Parallelism■ int add (int a, int b, int & result)■ // function stuff■ int sub(int a, int b, int & result) Data■ // function stuff Processor IS1 aa add add bpthread t1, t2; pthread t1, t2; bpthread-create(&t1, add, a,b, & r1); pthread-create(&t1, add, a,b, & r1); r1 r1 Processorpthread-create(&t2, sub, c,d, & r2); pthread-create(&t2, sub, c,d, & r2); c cpthread-par (2, t1, t2); IS2 pthread-par (2, t1, t2); sub sub dd r2 r2 MISD and MIMD Processing 342

343. Data Parallelism ■ sort(int *array, int count) Data ■ //..... Processor do ■ //..... “ Sort Sort “pthread-t, thread1, thread2;“ pthread-t, thread1, thread2; “ IS dn/2““pthread-create(& thread1, sort, array, N/2); pthread-create(& thread1, sort, array, N/2);pthread-create(& thread2, sort, array, N/2); pthread-create(& thread2, sort, array, N/2); Processorpthread-par(2, thread1, thread2); pthread-par(2, thread1, thread2); dn2/+1 Sort Sort “ “ SIMD Processing dn 343

344. Accessing Shared Resources■ Applications Access to Shared Resources need to be coordinated. ■ Printer (two person jobs cannot be printed at the same time) ■ Simultaneous operations on your bank account 344

345. Online Bank: Serving ManyCustomers and Operations PC client Internet Bank Server Local Area Network Bank Database PDA 345

346. Thread Priority■ In Java, each thread is assigned priority, which affects the order in which it is scheduled for running. The threads so far had same default priority (ORM_PRIORITY) and they are served using FCFS policy. ■ Java allows users to change priority: ■ ThreadName.setPriority(intNumber) ■ MIN_PRIORITY = 1 ■ NORM_PRIORITY=5 ■ MAX_PRIORITY=10 346

347. Thread Priority Exampleclass A extends Thread{ public void run() { System.out.println("Thread A started"); for(int i=1;i<=4;i++) { System.out.println("t From ThreadA: i= "+i); } System.out.println("Exit from A"); }}class B extends Thread{ public void run() { System.out.println("Thread B started"); for(int j=1;j<=4;j++) { System.out.println("t From ThreadB: j= "+j); } System.out.println("Exit from B"); }} 347

348. Thread Priority Exampleclass C extends Thread{ public void run() { System.out.println("Thread

C started"); for(int k=1;k<=4;k++) { System.out.println("t From ThreadC: k= "+k); } System.out.println("Exit from C"); }}class ThreadPriority{ public static void main(String args[]) { A threadA=new A(); B threadB=new B(); C threadC=new C(); threadC.setPriority(Thread.MAX_PRIORITY); threadB.setPriority(threadA.getPriority()+1); threadA.setPriority(Thread.MIN_PRIORITY); System.out.println("Started Thread A"); threadA.start(); System.out.println("Started Thread B"); threadB.start(); System.out.println("Started Thread C"); threadC.start(); System.out.println("End of main thread"); }} 348