if (one way selection) and if-else (Two ways selection) Statements The one-way if statement executes a certain block of code, ONLY if a particular condition is true if(Boolean-Expression) if(summax) statement; delta = sum - max; The two-way if-else statement chooses between two different execution paths.Inheritance o "is-a" relationship. o Allows us to specify relationships between types (classes). o Allows for code reuse. o Single Inheritance - Subclass (or child or derived) is derived from one existing class (superclass or parent) o Multiple Inheritance - Subclass is derived from more than one superclass. - Not supported by Java but supported in C++. - In Java, a class can only extend the definition of one class but could implement multiple interfaces (more later). Inheritance (Cont.) o Subclasses (child or derived class) have all of the data members and methods of the superclass (parent class). o General functionality can be written once and applied to *any* subclass(code reuse). o Subclasses are more specific and have more functionality. o Superclasses (parent class) capture generic functionality common across many types of objects. o Subclasses can specialize by: - adding data members. - adding methods. - overriding methods (more later). Inheritance (Cont.) o modifier(s) class ChildClassName extends ExistingParentClassName { membersList } o class Circle Derived from class Shape public class Circle extends Shape { . . . } Example: Shared Functionality Example: Shared Functionality (Cont.) Example: UML Diagram o What if we want to implement a 3d box object? o Both a Rectangle and a Box have a surface area, but they are computed differently. o Method Overriding! (more later!) Objects myRectangle and myBox Rectangle myRectangle = new Rectangle(5, 3); Box myBox = new Box(6, 5, 4); The Object Class and Its Methods o Every class in Java is descended from the java.lang.Object class. If no inheritance is specified when a class is defined, the superclass of the class is Object. public class Circle { ... } Equivalent public class Circle extends Object { ... } o toString() method in object class The toString() method returns a string representation of the object. - Protected members protected Access Modifier o The protected modifier creates a member that is accessible within its package and to subclasses in other packages. o protected modifier can be applied on data and methods in a class. Classes and Packages Used to Illustrate Access Levels Visibility Modifier Alpha same class Beta same Package AlphaSub subclass Gamma different package public Yes Yes Yes Yes protected Yes Yes Yes No no modifier (default) Yes Yes No No private Yes No No No The table shows where the members of the Alpha class are visible for each of the access modifiers that can be applied to them. Polymorphism Polymorphism o The dictionary definition of polymorphism refers to a principle in biology in which an organism or species can have many different forms or stages. o The simplest form of polymorphism (method overloading) means that the method will have more than one form (multiple implementations) with the same name but with different arguments (parameters). o Other form of polymorphism (method overriding). Polymorphism: Method Overloading o On the other hand, two methods cannot co-exist in the same class if they have the same name and arguments (parameters) and different return types. o For example, the two following methods will generate a compiler error . o The basic rule when creating overloaded methods is that every method must have a unique signature. o The parameters may differ in their types or numbers or in both. o They may have the same or different return types. Polymorphism: Method Overriding o A subclass inherits methods from a superclass. Sometimes it is necessary for the subclass to modify the implementation of a method defined in the

superclass. This is referred to as method overriding. o An instance method can be overridden only if it is accessible. Thus, a private method cannot be overridden, because it is not accessible outside its own class. If a method defined in a subclass is private in its superclass, the two methods are completely unrelated. o A static method can be inherited but can NOT be overridden. If a static method defined in the superclass is redefined in a subclass, the method defined in the superclass is hidden. o Recall: Instance methods are methods that require an object of its class to be created before it can be called (not static). Polymorphism: Method Overriding (Cont.) o A subclass can override (re-define) the methods of the superclass. o Objects of the subclass (child) type will use the new method (in the child class). o Objects of the superclass (parent) type will use the original method (in the parent class). Class Box public double area() { return 2 * (getLength() * getWidth() + getLength() * height + getWidth() * height); } Class Rectangle public double area() { return getLength() * getWidth(); } Overriding vs. Overloading public class Test { public static void main(String[] args) { A a = new A(); a.p(10); a.p(10.0); } } class B { public void p(double i) { System.out.println(i * 2); } } class A extends B { // This method overrides the method in B public void p(double i) { System.out.println(i); } } public class Test { public static void main(String[] args) { A a = new A(); a.p(10); a.p(10.0); } } class B { public void p(double i) { System.out.println(i * 2); } } class A extends B { // This method overloads the method in B public void p(int i) { System.out.println(i); } } Polymorphism (Cont.) o Can treat an object of a subclass as an object of its superclass. o Recall: A reference variable is one that refers to the address of another variable. o A reference variable of a superclass type can point to an object of its subclass. o A variable of a given type may be assigned a value of any subtype. o The reference variable name can point to any object of the class Person or the class PartTimeEmployee o name reference variable has many forms, that is, it is polymorphic reference variable. Person name= new Person(); PartTimeEmployee employee = new PartTimeEmployee(); name = employee; Polymorphism and References – Reference variables can refer to objects of their own class or to objects of the subclasses inherited from their class. Access superclass constructors and methods Calling methods of the superclass o To write a method's definition of a subclass, specify a call to the public method of the superclass. o If subclass overrides public method of superclass, specify call to public method of superclass: super.methodName(parameters list); – The keyword super refers to the superclass of the class in which super appears. This keyword can be used in two ways: 1. To call a superclass constructor 2. To call a superclass method o If subclass does not override public method of superclass, specify call to public method of superclass: methodName(parameters list); Calling methods of the superclass o Box class overloads the method setDimension() (Different parameters) Class Box public void setDimension(double l, double w, double h) { setDimension(l, w); if (h >= 0) height = h; else height = 0; } Defining Constructors of the Subclass o Superclass's constructors are not inherited in the subclass. o They can only be invoked from the subclasses' constructors, using the keyword super . Invoking a superclass constructor's name in a subclass causes a syntax error – Must be first statement – Specified by super parameter list. o If the keyword super is not explicitly used, the superclass's no-arg constructor is automatically invoked. public Box() { super(); height = 0; } public Box(double l, double w, double h) { super(l, w); height = h; } //Constructors in Java can be overloaded – Preventing class extending and method overriding final

Keyword o final methods – Can declare a method of a class as final using the keyword final. – If a method of a parent class is declared final, it cannot be overridden with a new definition in a derived class (subclass). o final class – Can also declare a class final using the keyword final. – final class can NOT be extended (no other class can be derived from this class). o final variable – The final variable is a constant. public final void doSomeThing() { //... } final class Math { ... } final static double PI = 3.14159; Dynamic and Static Binding Dynamic and Static Binding o Polymorphism: to assign multiple meanings to the same method name. o Implemented using – Method Overriding: Dynamic binding . – Method Overloading: Static binding o Binding – linking between method call and method definition. Static Binding vs. Dynamic Binding o Static Binding – Binding which can be resolved at compile time by compiler . – Also called early binding. – Binding happens before a program actually runs. – Example: Method Overloading. o Dynamic Binding – When compiler is not able to resolve the binding at compile time. – Also called late binding or run–time binding. – Binding happens during run time (execution time not compile time). – Example: Method Overriding Method Matching vs. Binding o Matching a method signature and binding a method implementation are two issues. The compiler finds a matching method according to parameter type, number of parameters, and order of the parameters at compilation time. A method may be implemented in several subclasses. The Java Virtual Machine dynamically binds the implementation of the method at runtime. Early/Static Binding Late/Dynamic Binding (at compile time) Late/Dynamic Binding (at run time) Static Binding vs. Dynamic Binding – Static binding uses Type information for binding while Dynamic binding uses objects to resolve binding. – Binding of all the static, private and final methods is done at compile– time. – Overloaded methods are resolved using static binding while overridden methods using dynamic binding i.e. at run time. Motivation o An interface is for defining common behavior for classes (including unrelated classes). Before discussing interfaces, we introduce a closely related subject: abstract classes. Abstract methods and classes o Suppose we plan to implement a number of shape classes: Rectangle, Square, Ellipse, Triangle, and so on. o We can give these shape classes out two basic area() and circumference() methods. o We want the Shape class to encapsulate whatever features all out shapes have in common (e.g. the area() and circumference() methods). But our generic Shape class doesn't represent any real kind of shape, so it cannot define useful implementation of the methods. o Java handles this situation with abstract methods. o An abstract method is a method header (method declaration) without a method body (method definition). Abstract methods and classes (Cont.) o Java let us define a method without implementing it by declaring the method with the abstract modifier . o An abstract class is a class that is declared with abstract modifier–– it may or may not include abstract methods. o An abstract class is mostly used to provide a base for subclasses to extend and implement the abstract methods and override or use the implemented methods in abstract class. o Rules about abstract methods and abstract classes that contain them: – Any class with an abstract method is automatically abstract itself and must be declared as such. – An abstract class cannot be instantiated using the new operator, but you can still define its constructors, which are invoked in the constructors of its subclasses. Abstract methods and classes (Cont.) – A subclass of an abstract class can be instantiated only if it overrides each of the abstract methods of its superclass and provides an implementation (i.e., a method body) for all of them. – In other words, in a

nonabstract subclass extended from an abstract class, all the abstract methods must be implemented, even if they are not used in the subclass. – static, private, and final methods cannot be abstract since these types of methods cannot be overridden by a subclass. Similarly, a final class cannot contain any abstract methods. – A subclass can be abstract even if its superclass is concrete. – A concrete class is a class that has an implementation for all of its methods. Abstract methods and class: Example (1/2) Abstract methods and class: Example (2/2) Interfaces o A Java interface is a collection of abstract methods and constants (finals). o An abstract method can be declared using the modifier abstract, but because all methods in an interface are abstract, usually it (abstract keyword) is left off. o An interface is used to establish a set of methods that a class will implement. (i.e. acts as a contract). Interfaces (Cont.) Interfaces (Cont.) o An interface cannot be instantiated (like abstract classes). o Methods in an interface are public (and abstract) by default. o A class formally implements an interface by: – stating so in the class header (... implements InterfaceName). – providing implementations for each abstract method in the interface. o If a class asserts that it implements an interface, it must define all methods in the interface. Interfaces (Cont.) Interfaces (Cont.) o A class that implements an interface can implement other methods as well. o In addition to, or instead of, abstract methods; an interface can contain constants (final). o When a class implements an interface, it gains access to all of its constants. Interfaces (Cont.) o In Java, a class can implement multiple interfaces but can inherit (extend) one class at maximum. o The interfaces (separated by commas) are listed in the implements clause. o The class must implement all the abstract methods in all interfaces listed in the header. o Multiple classes can implement the same interface Abstract Classes vs. Interfaces Interfaces Abstract Classes – Can have only public abstract methods i.e., by default – Can have protected, public and private abstract methods. – Can have only static final (constant) variable i.e., by default. – Can have static, final, or static final variable with any access specifier. – Constructor is not there. – Constructor is there. – No constructor, no object. – Constructor of abstract class not called directly (can be called using super). o Unified Modeling Language (UML) is a modeling language in the field of software engineering. The Unified Modeling Language (UML) includes a set of graphic notation techniques to create visual models of object–oriented software–intensive systems. (Wikipedia) Unified Modeling Language (UML) o Types of UML Diagrams: There are two main categories: 1) Structure Diagrams o Class Diagram o Package Diagram o Component Diagram o Deployment Diagram o Object Diagram o Profile Diagram o Composite Structure Diagram 2) Behavioral Diagrams o Use Case Diagram o Activity Diagram o State Machine Diagram o Sequence Diagram o Communication Diagram o Interaction Overview Diagram o Timing Diagram Class Diagram o What is a UML Class Diagram? Class diagrams are the main building block of any object–oriented solution. It shows the classes in a system, attributes, and operations of each class and the relationship between each class. o Basic Class Diagram Symbols and Notations Classes are illustrated with rectangles divided into three parts. 1– The name of the class in the first partition (centered, bolded, and first–letter capitalized), 2– The attributes in the second partition 3– The operations (methods) into the third. Class Diagram o Visibility Use visibility markers to signify who can access the information contained within a class. o Private visibility hides information from anything outside the class partition. o Public visibility allows all other classes to view the marked information. o Protected visibility allows child

classes to access information they inherited from a parent class. o Default visibility for package-level visibility. Class Diagram: Example o Static members o The UML denotes static features (attributes, methods) by underlining them. The static keyword may modify attributes and operations alike, and is independent of other modifiers such as public, private or protected. Class Diagram Class Diagram o Relationship(Association) o Relationship indicates that one class utilizes an attribute or method of another class. o The relationships between classes are drawn on class diagram by various lines and arrows. Inheritance Relationship o Inheritance: is represented with an empty arrow, pointing from the subclass (child class) to the superclass (parent class). bi-directional & uni-directional Relationship o If there is no arrow on the line, the association is taken to be bi-directional, that is, both classes hold information about the other class. o A uni-directional association is indicated by an arrow pointing from the object which holds to the object that is held. bi-directional & uni-directional association Aggregation Relationship o If the association conveys the information that one object is part of another object, but their lifetimes are independent (they could exist independently), this relationship is called aggregation. o For example, we may say that "a Department contains a set of Employees," or that "a Faculty contains a set of Teachers."Data Hiding Encapsulation is also known as "data Hiding" . Protects an object from unwanted access by clients. To achieve encapsulation in Java: Declare the variables of a class as private. Provide public setter and getter methods to modify and view the variables values. Data Hiding Real life examples of data hiding: 1) Your name and other personal information is stored in your brain we can't access this information directly. For getting this information we need to ask you about it and it will be up to you how much details you would like to share with us. 2) A bank application forbids (restrict) a client to change an account's balance. Getter and Setter Methods Get methods When an instance variable is declared private, we need a way to access that variable. A method used to obtain the value of a private instance variable is referred to as a get method (also known as getter or accessor methods). The naming scheme of getter should be as follows: public returnType getVariableName () { return variableName; } Get methods The method's return type specifies the type of data returned to a method's caller. Empty parentheses following a method name indicate that the method does not require any parameters to perform its task. The return statement passes a value from a called method back to its caller. To call a method of an object, follow the object name with a dot separator, the method name and a set of parentheses containing the method's arguments. Set Methods A method used to set the value of a private instance variable is referred to as a set method (also known as setter or mutator methods). The naming scheme of setter should as follows: public void setVariableName (Type local Variable) { VariableName = local Variable; } Keyword void indicates that a method will perform a task but will not return any information. Set methods A method call supplies values--known as arguments--for each of the method's parameters. Each argument's value is assigned to the corresponding parameter in the method header. The number of arguments in a method call must match the number of parameters in the method declaration's parameter list. The argument types in the method call must be consistent with the types of the corresponding parameters in the method's declaration. Get & Set methods By using getter and setter, the programmer can control how his important variables are accessed and updated in a correct manner, such as changing value of a variable within a specified range. Enumerations

Enumerated types o Enumerated types defines a new data type and list all possible values of that type enum Season {WINTER, SPRING, SUMMER, FALL} o Once established, the new type can be used to declare variables Season time; o The only values this variable can be assigned are the ones established in the enum definition o An enum is used for defining named constants values. Constant values that once it is defined, it cannot be added to, removed or changed just as a const variable. However, a Listis used to store objects too. Which can be manipulated such as add, remove and change the values in the list Enumerated types o An enumerated type definition is a special kind of class represents group of constants (unchangeable variables– final variables). o The values of the enumerated type are objects of that type o For example, FALL is an object of type Season o The following assignment is valid: time = Season.FALL; o But the following statement is not valid: time = new Season(); o Enumerated types are like classes, we can add additional instance data and methods. Passing and Returning Objects Passing Objects to a Method o As we can pass int and double values, we can also pass an object to a method. o When we pass an object, we are actually passing the reference (name) of an object o it means a duplicate of an object is NOT created in the called method Example We pass the same Student object to card1 and card2 o Since we are actually passing a reference to the same object, it results in owner of two LibraryCard objects pointing to the same Student object class LibraryCard { private Student owner; public void setOwner(Student st) { owner = st; } } class Student { private String name; private String email; public void setName(String name) { this.name = name; } public void setEmail(String email) { this.email = email; } } Finalize Method o finalize( ) method contains action to be performed to release memory resources held by the object before destroying it. o It is called by a garbage collector: System.gc(); o Garbage collector in Java is the automated process of deleting code that's no longer needed or used. This automatically frees up memory space and ideally makes coding Java apps easier for developers. o To add a finalizer to a class, define the finalize( ) method which is called by the Java run time whenever it is about to recycle an object of that class protected void finalize( ){ // finalization code here, for example: System.out.println("finalize method called"); } finalize( ) Method (Destructor) What is the best way to design these classes so to avoid redundancy?if (Boolean–Expression) if (number % 2 == 0) statement_1; System.out.print("Even"); else else statement_2; System.out.print ("Odd"); If the Boolean–Expression is true, statement_1 is executed; Otherwise, statement_2 is executed. The Nested if Statement – Multiway Selection o It is possible to choose between several actions (3 or more) using the nested if statement o Nested if: ? The statement inside an if or if–else statement can be any legal Java statement, including another if or if–else statement. The inner if statement is said to be nested inside the outer if statement ?The inner if statement can contain another if or if–else statement ?There is no limit to the depth of the nesting The switch–case Statement – Multiway Selection ?The switch statement is the only other kind of Java statement that implements multiway branching o When a switch statement is evaluated, one out of many different branches is executed o The choice of which branch to execute is specified by a controlling expression enclosed in parentheses after the keyword switch o Such expression must evaluate to the char, int, short, or byte data type switch (Controlling_Expression){ ...} o The controlling expression may consist of a single variable or an arithmetic expression, such as 1 + x * y The case Keyword ?Each branch in a switch statement starts

with the case keyword, followed by a constant called a case label, a colon (:), and then a sequence of statements o Each case label must be of the same type as the controlling expression o The order of the case labels does not matter but each one must appear only once. However, it is a good practice to follow the logical sequence of the labels o The code for a certain case label is executed when the value of that case label matches the value of the controlling expression o The case labels are constant expressions, meaning that they cannot contain variables in the expression, such as 1 + x o The case labels may contain a constant variable. Such a variable is declared with the keyword final The break keyword ?The break keyword is optional, but it should be used at the end of each case in order to skip the remainder of the switch statement ?The switch statement ends when it executes the break statement, or when the end of the switch statement is reached o When the computer executes the statements after a case label, it continues until a break statement is reached o If the break statement is deleted, then after executing the code for one case, the computer will go on to execute the code for the next case o If the break statement is deleted mistakeably, the compiler will not issue an error message The default Keyword ?There can also be a section labeled default: o The default section is optional, and is usually last o Even if the case labels cover all possible outcomes in a given switch statement, it is still a good practice to include a default section o It can be used to output an error message, for example ? When the controlling expression is evaluated, the code for the case label whose value matches the controlling expression is executed o If no case label matches, then the only statements executed are those following the default label (if there is one) Repetition (Loop) Statements o Allow us to execute a statement (or a block of statements) multiple times. They are contro led by boolean expressions o Keep repeating the loop body as long as the condition is true. When the condition becomes false, the repetition terminates, and the first statement after the loop body executes o Java has three types of repetition (loop) statements: o while statement: A pre-test loop that evaluates a condition before each loop o do-while statement: A post-test loop that evaluates a condition after each loop o for statement: A pre-test loop that evaluates a condition before each loop o Control of loop: o A control variable (or loop counter) o An initial value of the control variable o A loop condition that determines if looping should continue o An increment or a decrement expression that updates a control variable o The loop body that consists of one or more statements int count = 0; while (count100 ({ System.out.println("Welcome to Java"); count++; } The while Statement The while statement continua ly executes a block of code (i.e., the loop body) while a particular condition is true o The loop condition is first evaluated, which returns a boolean value o If the loop condition evaluates to false, the loop body will not execute and the control flow transfers to the statement after the loop body ?Thus, the loop body executes zero or more times (a pre-test loop) o If the loop condition evaluates to true, the loop body will execute o The while statement continues testing the loop condition and executing its body until the condition becomes false The do-while Statement The do-while statement continua ly executes a block of code (i.e., the loop body) while a particular condition is true o The loop body is first executed and then the loop condition is evaluated, which returns a boolean value o If the loop condition evaluates to false, the loop body will not execute again and the control flow transfers to the statement after the loop body ?Examples of Class OOP Concepts: (2) Object (Instance) o Objects (instances) basically are data structures combined with the

associated processing routines (methods).Example for array objects: – int[] myArray; // declaration – ?????? (This declares myArray to be an array of integers.Calling Method (Method Invocation