Inheritance information technology

Technology, Information Technology

Inheritance One of the major tenets of Object-Oriented programming is promotion of re-use. Most objects in real life have certain things in common for example a car, a truck and a motorcycle are all vehicles but with different distinguishing features such as size, wheels and maximum speed. In object-oriented program a good design practice calls for creation of fairly generic version of a class. The programmer then uses these generic classes to build up specialized classes. This is achieved through inheritance. Oracle (2012) defines inheritance as that ability of a class to inherit commonly used structure and behavior from another class. Continuing with the vehicle example it is clear that cars, trucks and motorcycles all have engines, wheels, shape, speed and other attributes which are expected of all vehicles. The subclass inherits from the superclass (Eck, 2011). In C++, the subclass is also referred to as the derived class or child class while the superclass is the base class or parent class. In Java, a class arrow points toward the class from which it extends. An example is shown in the diagram below. In object-oriented program class inheritance and interface implementation are used to manifest the concept of IS-A relationship. An IS-A relationship states that this item is a type of that item (Sierra & Bates, 2005). Using our example one can correctly say a car IS-A vehicle or a truck IS-A vehicle. In Java this unique relationship, IS-A relationship, is expressed through class inheritance and interfaces. To show inheritance the keyword extends is used whereas to show interface implementation the keyword implements is used. This is demonstrated below: class Vehicle { public void displayVehicle() { System. out. println(“ displaying vehicle”); } // additional code } // end Vehicle class Car extends Vehicle { int numberOfDoors; } // more code }// end Car class

Motorcycle extends Vehicle { int numberOfWheels; } // more code }// end Motorcycle class Truck extends Vehicle { int numberOfAxles; } // more code }// end Truck public class TestVehicles { public static void main (String[ ] args) { Car ferrari = new Car(); ferrari. displayVehicle(); ferrari. numberOfDoors(); }//end main }//end TestVehicles In the example above we see the element of code re-use where a program could refer to ferrari. numberOfDoors, an member variable of the Car class as well ferrari. displayVehicle() a method it inherits from the class Vehicle. In the above example the method displayVehicle() could be applied to a wide range of different kinds of vehicles without it having to be re-implemented. This means that all subclasses of Vehicle are guaranteed to have the behavior of their superclass Vehicle (Weiss, 2006). The second purpose of inheritance is to allow for classes to be accessed polymorphically. Suppose at the time of writing the Vehicle class, the programmer has no idea of the different forms of vehicles that could be written by any other programmer. However you may want a certain program that you write now to be able to call a method from any class that extends the Vehicle class. Polymorphism allows us to treat any subclass of Vehicle as a Vehicle. This means that the programmer can call, in our example the method displayVehicle() from any object that inherits / extends the class Vehicle. An example of using inheritance to allow for polymorphism of classes is shown below: class Vehicle { public void displayVehicle() { System. out. println(“ displaying vehicle”); } // more code } // end Vehicle class Car extends Vehicle { int numberOfDoors; } // more code }// end Car class Motorcycle extends Vehicle { int numberOfWheels; } // more code }// end Motorcycle Now imagine a test

class has a method with a declared argument type of Vehicle. This means any subclass of Vehicle can be passed to the method with argument of type Vehicle. This code is shown below: public class TestVehicles { public static void main (String[ ] args) { Car ferrari = new Car(); Motorcycle Honda = new Motorcycle(); showVehicle(ferrari); //output is displaying vehicle showVehicle(honda); //output is displaying vehicle }//end main public static void showVehicle(Vehicle vessel){ vessel. displayVehicle(); } }//end TestVehicles The output from the above code is the same. The important point to note is that the methods called on a reference are totally dependent on the declared type of the variable, no matter what the actual object is, that the reference is referring to (Sierra & Bates, 2008). There is one major difference of inheritance between Java and other programming languages such as C++. In Java, a class cannot inherit more than one class. In C++ a class can have multiple inheritances. For example in C++ class A can extend B and C at the same time as shown below class A extends B, C { // more code } // this is not possible in Java Java prohibits this because of the so called" Deadly Diamond of Death". In the diagram shown below suppose Vessel extends Car and Motorcycle each of which has a start() method which of these versions will the Vessel subclass inherit? References Eck, D. J. (2011). Introduction to Programming Using Java, Version 6. 0. Retrieved from http://math. hws. edu/javanotes/ Oracle. (2012). What Is Inheritance? The JavaTM Tutorials. Retrieved August 8, 2012, from http://docs. oracle. com/javase/tutorial/java/concepts/inheritance. html Sierra, K., & Bates, B. (2005). Head First Java (2nd ed.). Sebastopol, CA: O'Reilly Media, Inc. Sierra, K., & Bates, B. (2008). SCJP Sun Certified Programmer for Java 6 Study Guide

Exam (310-065). New York: McGraw-Hill. Weiss, M. A. (2006). Data Structures & Problem Solving Using Java (3rd ed.). Boston, MA: Pearson Education.