* In OO design, the attributes and behaviors are contained within a single object, whereas in procedural or structured design, the attributes and behaviors are normally separated.
* True
* In OO, you can reuse a code as many times as you want
* True
* A subclass inherits attributes from its superclass, which allows it to perform as a subclass
* + True
* There are only two ways to build classes from other classes: inheritance and polymorphism
* + False
* Java allows an instance of an abstract class to be instantiated
* + False
* Early binding enables the compiler to be more efficient.
* + True
* If the final modifier is included before the definition of the method, then the method can be redefined in a derived class.
* + False
* You cannot derive an interface from a base interface
* + False
* The throw operator causes a change in the flor control
* + True
* When an exception is thrown, the code in the surrounding try black continues executing and then the catch block begins executing.
* + False
* Inheritance and composition have one significant difference in the way objects are built. When inheritance is used, the end result is a single class that incorporates all the behaviors and attributes of the inheritance hierarchy. When composition is used, one or more classes are used to build another class.
* + True
* An interface specifies the headings and definitions for methods that must be defined in any class that implements the interface.
* + False
* An abstract method serves as a placeholder for a method that muse be defined in all derived classes.
* + True
* Java source code that contains a class with an inner class, when compiled, will produce a separate .class file for the inner class.
* + True
* In a static method, you may use the this parameter either explicitly or implicitly.
* + False
* Generalization-specialization is the idea that as you make your way down the inheritance tree, things get more specific.
* + True
* Wrapper classes provide a class type corresponding to each of the primitive types so that you can have class types that behave somewhat like primitive types.
* + True
* In most cases, there is no reason to build a class if it is nog going to interact with other classes
* + True
* The String class is a mutable class
* + False
* An inner class definition is local to the outer class definition.
* + True
* Reusable classes tend to have
* + Interfaces that are more abstract than concrete
* What is the most important function of a constructor?
* + To allocate memory when the new keyword is encountered
* When private methods are meant to be part of the implementation and not the public interface, the methods are invoked
* + From within the class
* When you use the assignment operator with variables of a class type, you are assigning a
* + Reference
* The correct syntax for accessing the length of an array named numbers is:
* + numbers.length
* \_\_\_\_\_ refers to the process of associating a method definition with a method invocation
* + Binding
* A method or instance variable modified by protected:
* + Cannot be accessed by name in any other class (i.e, other than classes named in a-c.).
* Assigning an object of a derived class to a variable of a base class is called:
* + Upcasting
* A class with no abstract methods is called a
* + Concrete class
* ArrayIndexOutOfBoundsException is a descendent of the class RuntimeException. This means:
* + The exception does not have to be explicitly caught
* When used with objects, what is the equality (==) operator comparing?
* + == compares object references/ checks that two objects have the same memory address.
* Explain what a call to super( ) call does in a constructor of a derived class.
* + The super() calls a superclass default constructor.
* What is responsibility of Garbage Collector? What is purpose of overriding finalize() method?
* + Garbage Collector reclaims memory used by an object
  + Overriding finalize() when class has resources that won’t be cleaned up by the Garbage Collector.
* What is the output of the following program?

class Base{

protected void foo() {System.out.println(“Base Foo called “); }

}

class Derived extend Base {

void foo() { System.out.println(“Derived Foo called ”); }

}

public class Main {

public static void main(String args[]) {

Derived d = new Derived();

d.foo();

}

}

* Compilation error. Foo in class Base is protected and attempting to override it in Derived will not work.
* What is the output of the following program? Explain

public class StringOper{

public static void main(String [] args)

{

String s1 = “abc”;

String s2 = “def”;

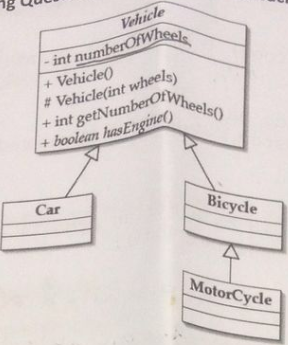
String s3 = s2;

s2 = “ghi”;

System.out.println(s1 + s2 +s3);

}

}

* abcghidef
* Before system has a chance to print out, s2’s original memory is changed.
* If you were going to catch an exception of type Exception along with catching other exceptions, where should this catch block be placed within your code?
* + The catch box should be placed at the last to avoid interruption of the normal flow caused by exception. If put ( catch (exception e)) first, then in case of any exception this block will execute.
* 

Consider the class hierarchy of vehicles shown in the Figure above, with the following additional information:

* Cars always have four wheels;
* Bicycles and their subclasses always have two wheels.

**Note**: there are only default (i.e., parameter-less) constructors available in the subclasses.

2. Convert the class hierarchy in Figure 1 to Java. Provide implementation to all the methods.
4. Write the implementation of the main() method as a test drive to instantiate an object of each class type and sore all the instantiated objects in a single VehiclesArray.
6. For each of the following statements pertinent to classes of the hierarchy state whether they are acceptable or not in Java and give a brief (no more than two lines) explanation why it is or is not the case.
8. Vehicle v1 = new Vehicle();
10. Vehicle [] v2 = new Vehicle[3];
12. Vehicle v3 = new MotorCycle();
14. Bicycle b = new MotorCycle();
16. //Car c1 = new Bicycle ();
18. Vehicle v4 = new MotorCycle();
20. Car c2 = (Car) v4;
22. Vehicle v5 = new Car();
24. Car c3 = (Car) v5;

public class Vehicle {

private int numberOfWheels;

public vehicle();

protected Vehicle(int wheels);

public GetNumberOfWheels(int);

abstract hasEngine(boolean);

Vehicle (){

}

protected Vehicle(int wheels){

}

int getNumberOfWheels(){

return numberOfWheels;

}

public abstract class boolean hasEngine(){

return hasEngine;

}

}

class Car extends Vehicle{

final int numberOfWheels = 4;

}

class Bicycle extends Vehicle{

final int numberOfWheels = 2;

}

class MotorCycle extends Bicycle{

final int numberOfWheels = 2;

}

public static void main(String[] args) {

Vehicle v1 = new Vehicle(); - acceptable Because v1 is an object of class Vehicle and it is instantiating it giving it a new memory location.

Vehicle [] v2 = new Vehicle[3]; - acceptable it is creating array of an object v2 and then giving the array size of 3.

Vehicle v3 = new MotorCycle(); - acceptable, similar to v1, however it is giving the new memory location to inside of MotorCycle.

Bicycle b = new MotorCycle(); - acceptable similar to v1, however it is giving the new memory location to inside of MotorCycle from bicycle

Car c1 = new Bicycle(); - not acceptable bicycle cannot be converted to car

Vehicle v4 = new MotorCycle(); Car c2 = (Car) v4; - not acceptable MotorCycle cannot be cast to car

Vehicle v5 = new Car(); Car c3 = (Car) v5; - acceptable car can cast to car

}