OOP: True or False

Every object in Java is an Object.

The “is-a” relation is implemented by inheritance.

The “has-a” relation is implemented by the keyword **extends**.

An object of a subclass contains copies of all of the instance variables of its parent class, grandparent class, and so on.

A subclass may not re-declare instance variables from its superclass.

A subclass may not redefine instance methods in its superclass.

Although a class may have many subclasses, it may have only one direct superclass.

When an instance method is called, say *a.meth()*, the method executed depends on the type of the object referred to by *a* and not on the type of the variable *a*.

A variable of a class may be assigned an object from any of its subclasses.

A variable of a superclass may be downcast to any of its subclasses.

If the method *toString()* is overridden in a class, it must be declared public.

If B is a subclass of A, then a B object may always be assigned to a variable of type A.

If A and its subclass B both have a method called *compute* and *a* is a variable of type A, then *a.compute()* always calls the method in the superclass.

A method *toString(String s)* defined in a class overrides the *toString* method in Object.

Upcasting requires an explicit casting operation.

Downcasting requires an explicit casting operation, but is always legal.

Polymorphism works in Java because method calls are bound to their method definitions at compile time.

The principle of overloading is what makes polymorphism work in Java.

Calling polymorphic method *meth* that is shared by a collection of subclasses of some class A, a subclass of Object, requires that the method be declared in A as well.

If class A extends class B, class A is a subclass of B and B is a superclass of A.

An instance (an object) of a superclass can be treated as if it were an instance of any of its subclasses.

Polymorphism implies that the method actually called at runtime depends on the class of the method’s parameters.

Classes in Java may extend up to two other classes.

Downcasting an object to one of its subclasses may cause a compiler error.

Since interfaces cannot be instantiated, we cannot assign an object to an interface variable.

A class method may be called directly from inside an instance method in the same class.

An instance method may be called directly from inside a class method.

Instance methods always have an implicit parameter.

A class method may not be called from outside its class.

Class methods have an implicit parameter.

Mutator methods are procedures and accessors are functions.

A method cannot have both an implicit parameter and an explicit one.

Interfaces

A class in Java may implement only one interface.

An interface has no bodies (definitions) for its methods, which are automatically public.

An object of a class that implements an interface may be upcast to that interface type.

A class may extend only one other class and implement only one interface.

An abstract class that implements an interface must provide code for all methods in the interface.

A variable whose type is an interface may be assigned any object whose class implements the interface.

Polymorphism can be applied to a collection of classes that implement a common interface.

A class that contains an abstract method must be declared abstract.

A method inside an abstract class must be declared abstract.

A class may implement more than one interface.

Methods

Two formal parameters for the same method may use the same name.

The value of the implicit parameter to an instance method may not be accessed inside of the method.

Overridden methods are distinguished by their parameters.

Overloaded methods are distinguished by their parameters.

A method may not return an array of objects as its result.

An array passed as a parameter to a method may be altered from inside the method.

To sort an array in a method, both it and its size must be passed in as parameters.

Constructors

Constructors must have parameters.

Java provides a parameterless constructor for each class.

Objects cannot be created without applying the **new** operator.

The compiler reports an error if we create object with a class for which we have not written a constructor.

A constructor is more like a class method than an instance method.

If we do not make a call of super() (with or without parameters) in a constructor, it is done automatically.

Variables

Each object of a class has copies of all of the instance and class variables of the class.

Variables declared inside of a method are called instance variables.

Instance variables must be declared either public or private.

Local variables must be given values explicitly before they are used.

A class may not have both instance variables and class variables.

A local variable may not be accessed from outside of its method.

Instance variables in a class may be referred to from another class.

Instance variables in a class must be private.

Any variable declared with a class name as its type may be assigned the value **null**.

A **char** value may always be assigned to an **int** variable.

Programming

A String object may not have its characters altered.

Two methods may share the same name.

Putting comments in your code will slow down your program.

Loops of any kind can be nested only up to three levels deep.

The body of a while loop will be executed at least once.

The error message “ArrayIndexOutOfBound” is generated by the compiler.

Classes and Objects

If d refers to a Doodad object, then

```
System.out.println("d = " + d);
```

will produce output on the screen.

A class must have at least one instance variable.

Every object in Java is an Object.

When an object is created using **new**, it must be assigned to a variable.

Class names in Java must begin with an uppercase letter.

Every object in Java will respond to the *toString* method.

To follow the principles of data encapsulation, we should declare all instance variables to be public.