A class is a template for an object, and an object is an instance of a class.

A class creates a new data type that can be used to create objects.

When you declare an object of a class, you are creating an instance of that class.

Thus, a class is a logical construct. An object has physical reality. (That is, an object occupies space in memory.)

Objects are characterized by three essential properties: state, identity, and behavior.

The state of an object is a value from its data type. The identity of an object distinguishes one object from another.

It is useful to think of an object's identity as the place where its value is stored in memory.

The behavior of an object is the effect of data-type operations.

The dot operator links the name of the object with the name of an instance variable.

Although commonly referred to as the dot operator, the formal specification for Java categorizes the . as a separator.

The 'new' keyword dynamically allocates(that is, allocates at run time)memory for an object & returns a reference to it.

This reference is, more or less, the address in memory of the object allocated by new.

This reference is then stored in the variable.

Thus, in Java, all class objects must be dynamically allocated.

```
Box mybox; // declare reference to object
    mybox = new Box(); // allocate a Box object
```

The first line declares mybox as a reference to an object of type Box. At this point, mybox does not yet refer to an

actual object. The next line allocates an object and assigns a reference to it to mybox. After the second line executes,

you can use mybox as if it were a Box object. But in reality, mybox simply holds, in essence, the memory address of the

actual Box object.

The key to Java's safety is that you cannot manipulate references as you can actual pointers.

Thus, you cannot cause an object reference to point to an arbitrary memory location or manipulate it like an integer.

A Closer Look at new:

classname class-var = new classname ( );

Here, class-var is a variable of the class type being created. The classname is the name of the class that is being

instantiated. The class name followed by parentheses specifies the constructor for the class. A constructor defines

what occurs when an object of a class is created.

You might be wondering why you do not need to use new for such things as integers or characters.

The answer is that Java's primitive types are not implemented as objects.

Rather, they are implemented as "normal" variables.

This is done in the interest of efficiency.

It is important to understand that new allocates memory for an object during run time.

Box b1 = new Box();

Box b2 = b1;

b1 and b2 will both refer to the same object. The assignment of b1 to b2 did not allocate any memory or copy any part

of the original object. It simply makes b2 refer to the same object as does b1. Thus, any changes made to the object

through b2 will affect the object to which b1 is referring, since they are the same object.

When you assign one object reference variable to another object reference variable, you are not creating a copy of the

object, you are only making a copy of the reference.

```
int square(int i){
   return i * i;
}
```

A parameter is a variable defined by a method that receives a value when the method is called. For example,

in square(int i), i is a parameter. An argument is a value that is passed to a method when it is invoked.

For example, square(100) passes 100 as an argument. Inside square( ), the parameter i receives that value.

NOTE:

Bus bus = new Bus();

lhs(reference i.e. bus) is looked by compiler & rhs (object i.e. new Bus()) is looked by jvm