Quiz On Chapter 5

**Question 1:**

Object-oriented programming uses *classes* and *objects*. What are classes and what are objects? What is the relationship between classes and objects?

**Answer:**

When used in object-oriented programming, a class is a factory for creating objects. (We are talking here about the non-static part of the class.) An object is a collection of data and behaviors that represent some entity (real or abstract). A class defines the structure and behaviors of all entities of a given type. An object is one particular "instance" of that type of entity. For example, if *Dog* is a class, then a particular dog named *Lassie* would be an object of type *Dog*.

**Question 2:**

Explain carefully what *null* means in Java, and why this special value is necessary.

**Answer:**

When a variable is of object type (that is, declared with a class or interface as its type rather than one of Java's primitive types), the value stored in the variable is not an object. Objects exist in a part of memory called the heap, and the variable holds a *pointer* or *reference* to the object. Null is a special value that can be stored in a variable to indicate that it does not actually point to any object.

**Question 3:**

What is a *constructor?* What is the purpose of a constructor in a class?

**Answer:**

A constructor is a special kind of subroutine in a class. It has the same name as the name of the class, and it has no return type, not even void. A constructor is called with the new operator in order to create a new object. Its main purpose is to initialize the newly created object, but in fact, it can do anything that the programmer wants it to do.

**Question 4:**

Suppose that Kumquat is the name of a class and that fruit is a variable of type Kumquat. What is the meaning of the statement "fruit = new Kumquat();"? That is, what does the computer do when it executes this statement? (Try to give a complete answer. The computer does several things.)

**Answer:**

This statement creates a new object belonging to the class Kumquat, and it stores a reference to that object in the variable fruit. More specifically, when the computer executes this statement, it allocates memory to hold a new object of type Kumquat. It calls a constructor, which can initialize the instance variables of the object as well as perform other tasks. A reference to the new object is returned as the value of the expression "new Kumquat()". Finally, the assignment statement stores the reference in the variable, fruit. So, fruit can now be used to access the new object.

**Question 5:**

What is meant by the terms *instance variable* and *instance method*?

**Answer:**

Instance variables and instance methods are non-static variables and methods in a class; that is, their definitions in the class are **not** marked with the "static" modifier. This means that they do not belong to the class itself. Instead, they specify what variables and methods are in an object that belongs to that class. That is, the class contains the source code that defines instance variables and instance methods, but actual instance variables and instance methods are contained in objects, at least logically. (Such objects are called "instances" of the class.) Thus, instance variables and instance methods are the data and the behaviors of objects.

**Question 6:**

Explain what is meant by the terms *subclass* and *superclass.*

**Answer:**

In object oriented programming, one class can inherit all the properties and behaviors from another class. It can then add to and modify what it inherits. The class that inherits is called a subclass, and the class that it inherits from is said to be its superclass. In Java, the fact that ClassA is a subclass of ClassB is indicated in the definition of ClassA as follows:

class ClassA extends ClassB {...}

**Question 7:**

Modify the following class so that the two instance variables are private and there is a getter method and a setter method for each instance variable:

public class Player {

String name;

int score;

}

**Answer:**

To make a variable private, just add the word "private" in front of each declaration. We need two methods for each variable. One of them returns the value of the variable. The other provides a new value for the variable. The names for these methods should follow the usual naming convention for getter and setter methods. (Note that my setter methods use the special variable this so that I can use the same name for the parameter of the method as is used for the instance variable. This is a very common pattern.)

public class Player {

private String name;

private int score;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name; // ("this.name" refers to the instance variable)

}

public int getScore() {

return score;

}

public void setScore(int score) {

this.score = score;

}

}

**Question 8:**

Explain why the class *Player* that is defined in the previous question has an instance method named toString(), even though no definition of this method appears in the definition of the class.

**Answer:**

If a class is not declared to extend any class, then it automatically extends the class *Object*, which is one of the built-in classes of Java. So in this case, *Player* is a direct subclass of *Object*. The *Object* class defines a toString() method, and the *Player* class inherits this toString() method from *Object*. The methods and member variables in a class include not just those defined in the class but also those inherited from its superclass. (However, the inherited toString() method will not produce a useful string representation of a *Player*; to get that, you would have to override toString() in the *Player* class.)

**Question 9:**

Explain the term *polymorphism.*

**Answer:**

Polymorphism refers to the fact that different objects can respond to the same method in different ways, depending on the actual type of the object. This can occur because a method can be overridden in a subclass. In that case, objects belonging to the subclass will respond to the method differently from objects belonging to the superclass.

(Note: If B is a subclass of A, then a variable of type A can refer to either an object of type A or an object of type B. Let's say that var is such a variable and that action() is a method in class A that is redefined in class B. Consider the statement "var.action()". Does this execute the method from class A or the method from class B? The answer is that there is no way to tell! The answer depends on what type of object var refers to, a class A object or a class B object. The method executed by var.action() depends on the actual type of the object that var refers to **at run time**, not on the type of the variable var. This is the real meaning of polymorphism.)

**Question 10:**

Java uses "garbage collection" for memory management. Explain what is meant here by garbage collection. What is the alternative to garbage collection?

**Answer:**

The purpose of garbage collection is to identify objects that can no longer be used, and to dispose of such objects and reclaim the memory space that they occupy. If garbage collection is not used, then the programmer must be responsible for keeping track of which objects are still in use and disposing of objects when they are no longer needed. If the programmer makes a mistake, then there is a "memory leak," which might gradually fill up memory with useless objects until the program crashes for lack of memory.

**Question 11:**

What is an *abstract class*, and how can you recognize an abstract class in Java?

**Answer:**

An abstract class is one that cannot be used to create objects. It exists only as a basis for making subclasses, and it expresses all the properties and behaviors that those subclasses have in common. In Java, a class can be marked with the modifier abstract to make it abstract. For example,

abstract public class Vehicle { ...

It will then be a syntax error to try to call a "new Vehicle" constructor. (Note: Only a class that has been marked as abstract can contain abstract instance methods.)

**Question 12:**

What is this?

**Answer:**

"this" is a special variable in Java, which does not have to be declared. Java makes it available automatically in instance methods and constructors. It holds a reference to the object that is being constructed or that contains the instance method that is being executed (or, in terms of messages, the object that received the message that is being processed). It provides a way to refer to "this object." If x is an instance variable, it can also be referred to as this.x within the same class. If doSomething() is an instance method, it can also be called as this.doSomething() within the same class. (Personally, I would be happier with Java if it required the use of "this" instead of using it implicitly.)

**Question 13:**

For this problem, you should write a very simple but complete class. The class represents a counter that counts 0, 1, 2, 3, 4, .... The name of the class should be Counter. It has one private instance variable representing the value of the counter. It has two instance methods: increment() adds one to the counter value, and getValue() returns the current counter value. Write a complete definition for the class, Counter.

**Answer:**

Here is a possible answer. (Note that the initialization of the instance variable, value, to zero is not really necessary, since it would be initialized to zero anyway if no explicit initialization were provided.)

/\*\*

\* An object of this class represents a counter that counts up from zero.

\*/

public class Counter {

private int value = 0; // Current value of the counter.

/\*\*

\* Add one to the value of the counter.

\*/

public void increment() {

value++;

}

/\*\*

\* Returns the current value of the counter.

\*/

public int getValue() {

return value;

}

} // end of class Counter

**Question 14:**

This problem uses the Counter class from the previous question. The following program segment is meant to simulate tossing a coin 100 times. It should use two Counter objects, headCount and tailCount, to count the number of heads and the number of tails. Fill in the blanks so that it will do so:

Counter headCount, tailCount;

tailCount = new Counter();

headCount = new Counter();

for ( int flip = 0; flip < 100; flip++ ) {

if (Math.random() < 0.5) // There's a 50/50 chance that this is true.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ; // Count a "head".

else

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ; // Count a "tail".

}

System.out.println("There were " + \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ + " heads.");

System.out.println("There were " + \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ + " tails.");

**Answer:**

The variable headCount is a variable of type Counter, so the only thing that you can do with it is call the instance methods headCount.increment() and headCount.getValue(). Call headCount.increment() to add one to the counter. Call headCount.getValue() to discover the current value of the counter. Note that you can't get at the value of the counter directly, since the variable that holds the value is a private instance variable in the Counter object. Similarly for tailCount. Here is the program with calls to these instance methods filled in:

Counter headCount, tailCount;

tailCount = new Counter();

headCount = new Counter();

for ( int flip = 0; flip < 100; flip++ ) {

if (Math.random() < 0.5) // There's a 50/50 chance that this is true.

headCount.increment() ; // Count a "head", using headCount

else

tailCount.increment() ; // Count a "tail", using tailCount

}

System.out.println(("There were " + headCount.getValue() + " heads.");

System.out.println(("There were " + tailCount.getValue() + " tails.");

**Question 15:**

Explain why it can **never** make sense to test "if (obj.equals(null))".

**Answer:**

If the value of obj is **not** null, then the test correctly returns false. However, if the value of obj **is** null, then there is no such thing as obj.equals, and the attempt to evaluate obj.equals(null) will cause a *NullPointerException* that will crash the program! The correct way to test if the value of obj is null is "if (obj == null)".