1. Which members of the Circle class are encapsulated?

Encapsulated members are those that are declared private within the class. In the Circle class, any variables or methods marked as private are encapsulated. This means they can't be accessed directly from outside the class—only through public methods (like getters and setters). For example, if radius is declared as private double radius;, it is encapsulated.

2. What name must the constructor of a class have?

A constructor must have the same name as the class itself. For example, if the class is named Circle, then its constructor must also be named Circle(). This rule lets the compiler know which method is used to create (or "construct") an object of that class.

3. Explain the difference between the private and public access modifiers.

The private access modifier means that the variable or method can only be accessed within the same class.

The public access modifier allows the variable or method to be accessed from anywhere in the program, including outside the class.
Essentially, private keeps data safe and hidden, while public allows open access.

4. Consider the following code. Is the last statement valid or invalid? Explain.

```
Circle dot = new Circle(2);
dot.radius = 5;
```

This statement is invalid if radius is declared as private inside the Circle class. Private members cannot be accessed directly using the object name outside the class. You would need a public setter method, such as dot.setRadius(5);, to modify it. If radius were public, then it would be valid—but that would break encapsulation principles.

5. Use the following class to answer the questions below:

```
public class Roo {
   private int x;
   public Roo() {
      x = 1;
   }
   public void setX(int z) {
      x = z;
   }
   public int getX() {
      return x;
   }
```

```
    public int calculate() {
        x = x * factor();
        return x;
    }
    private int factor() {
        return 0.12;
    }
}
```

(Note: I corrected a few syntax errors — added missing parentheses and return types.)

a) What is the name of the class?
→ The name of the class is Roo.

b) What is the name of the data member?
→ The data member is x.

c) List the accessor method.
→ The accessor method is getX(), since it returns the value of x.

d) List the modifier method.
→ The modifier (or mutator) method is setX(int z), since it changes the value of x.

e) List the helper method.
→ The helper method is factor(), since it's private and used only within the class to assist another method.

f) What is the name of the constructor?
→ The constructor is Roo(), which has the same name as the class.

g) How many method members are there?
→ There are 5 methods in total: Roo(), setX(), getX(), calculate(), and factor().

6. What is the difference between a class and an object?

A class is like a blueprint or template that defines the structure and behavior (methods and variables) of something.
An object is a specific instance of that class, created in memory using the new keyword.
For example:

```
Circle ball = new Circle(5);
```

Here, Circle is the class, and ball is an object (an instance of Circle).

9. Use the following class data member definitions to answer the questions below:

```
public class Moo {
    private double y;
    private static int x;
    private static final z;
}
```

y is a private instance variable, meaning each object of Moo has its own copy.

x is a private static variable, meaning it belongs to the class itself, not to individual objects.

z is a private static final variable, meaning it's a class-level constant — its value cannot change once it's assigned.

11. Compare and contrast overriding methods to overloading methods.

Overriding happens when a subclass redefines a method that already exists in its superclass with the same name, parameters, and return type — but provides a new version of its behavior. Overloading happens when multiple methods in the same class share the same name but have different parameters (different number or type of arguments).

Example:

Overloading:

```
public void print(int x) {}
public void print(String s) {}
```

Overriding:

```
class Animal { void speak() {} }
class Dog extends Animal { void speak() { System.out.println("Bark"); } }
```

Overriding = different classes, same signature.
Overloading = same class, different parameters.