Which members of the Circle class are encapsulated?

In an object-oriented design, **encapsulation** refers to restricting access to the internal state of an object and providing access via public methods. Assuming the `Circle` class follows this principle, the **radius** and other internal attributes (like diameter, area, etc.) would be encapsulated if they are declared as **private** and accessed via **public getter and setter methods**.

For example:

```
public class Circle {
   private double radius; // Encapsulated member

   public double getRadius() {
      return radius;
   }

   public void setRadius(double radius) {
      this.radius = radius;
   }
}
```

1. Here, the **radius** is encapsulated by making it private, and its value is accessed and modified through public methods (`getRadius()` and `setRadius()`).

What name must the constructor of a class have?

The **constructor** of a class must have the **same name** as the class itself. Constructors do not have a return type, not even `void`. They are used to initialize objects when they are created.

Example:

```
public class Circle {
   private double radius;

   // Constructor name matches the class name
   public Circle(double radius) {
      this.radius = radius;
   }
}
```

2.

---

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

Private: A member (variable, method, or inner class) marked as **private** is only accessible within the class in which it is defined. It is not accessible from outside the class or by instances of other classes.

Example:

```
public class Circle {
   private double radius;  // Only accessible within this class
}
```

-

Public: A member marked as **public** is accessible from anywhere in the program, meaning other classes can access the public member directly.

Example:

```
public class Circle {
   public double radius;  // Accessible from anywhere
}
```

-

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

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

4. **Answer: The last statement is invalid.**

   Explanation:

   - If the `radius` variable in the `Circle` class is declared as **private**, it cannot be accessed directly outside the `Circle` class. So, trying to assign a value to `dot.radius` would result in a compilation error, because `radius` is private and cannot be accessed from outside the class.
   - To make this valid, you would need to provide a public setter method to allow modification of the `radius` value.

For example, you could modify the `Circle` class like this:

```
 public class Circle {
    private double radius;

    // Constructor
    public Circle(double radius) {
        this.radius = radius;
    }

    // Setter method to modify radius
    public void setRadius(double radius) {
        this.radius = radius;
    }

    // Getter method to retrieve radius
    public double getRadius() {
        return radius;
    }
}
```

Then, the statement would be valid if modified as:

```
 Circle dot = new Circle(2);
dot.setRadius(5);  // Use the setter method to change radius
```

The **difference between a class and an object** can be explained as follows:

1. **Class**:
    - A **class** is a blueprint or template for creating objects. It defines the properties (attributes) and behaviors (methods) that the objects of that class will have.
    - A class does not represent an actual entity but is rather an abstraction.
    - It specifies what an object of that class will contain and what actions it can perform, but it does not itself store any data or perform any actions.

    Example:

    ```
     public class Car {

        // Attributes (fields)

        private String color;
    ```

```
    private String model;


    // Behavior (method)
    public void startEngine() {
        System.out.println("Engine started");
    }
}
```

2. Here, `Car` is the class that defines a car with a `color` and `model` and the ability to `startEngine`.

3. **Object**:

    - An **object** is an instance of a class. It is a concrete entity that exists in memory when the class is instantiated.
    - An object holds the actual data (values for attributes) and can invoke the methods defined in the class to perform actions.
    - While a class is like a blueprint, an object is the actual "thing" created from that blueprint.

    Example:

```
Car myCar = new Car(); // myCar is an object of the Car class
myCar.color = "Red";    // Assign value to the attribute
myCar.startEngine();    // Call a method on the object
```

4. In this example, `myCar` is an object of the `Car` class, and it holds specific values for `color` and `model`.

Summary:

- **Class**: Defines the structure and behavior.
- **Object**: A specific instance of that class with its own data.

The **difference between a class and an object** can be explained as follows:

6. Class:

- A **class** is a blueprint or template for creating objects. It defines the properties (attributes) and behaviors (methods) that the objects of that class will have.
- A class does not represent an actual entity but is rather an abstraction.
- It specifies what an object of that class will contain and what actions it can perform, but it does not itself store any data or perform any actions.

Example:

```
public class Car {

    // Attributes (fields)

    private String color;

    private String model;


    // Behavior (method)

    public void startEngine() {

        System.out.println("Engine started");

    }

}
```

Here, `Car` is the class that defines a car with a `color` and `model` and the ability to `startEngine`.

Object:

- An **object** is an instance of a class. It is a concrete entity that exists in memory when the class is instantiated.

    - An object holds the actual data (values for attributes) and can invoke the methods defined in the class to perform actions.
    - While a class is like a blueprint, an object is the actual "thing" created from that blueprint.

Example:

```
Car myCar = new Car(); // myCar is an object of the Car class

myCar.color = "Red";    // Assign value to the attribute

myCar.startEngine();    // Call a method on the object
```

In this example, `myCar` is an object of the `Car` class, and it holds specific values for `color` and `model`.

Summary:

- **Class**: Defines the structure and behavior.
- **Object**: A specific instance of that class with its own data.