OOP Exercises in Java

Question 1: Constructors

What will the following code output?

java Copy code

```java
class Car {
    String model;

    Car(String model) {
        this.model = model;
    }

    Car() {
        this("Default Model");
    }
}

public class Main {
    public static void main(String[] args) {
        Car car = new Car();
        System.out.println(car.model);
    }
}
```

1. `null`
2. `Default Model`
3. Compilation Error
4. Runtime Exception

Question 2: Method Overriding

Consider the following code:

```java
class Animal {
    void speak() {
        System.out.println("Animal speaks");
    }
}

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

public class Main {
    public static void main(String[] args) {
        Animal animal = new Dog();
        animal.speak();
    }
}
```

What will the output be?

1. `Animal speaks`
2. `Dog barks`
3. **Compilation Error**
4. **Runtime Exception**

Question 3: Abstract Classes

Which of the following is **true** about abstract classes?

1. An abstract class can have both abstract and concrete methods.
2. Abstract classes can be instantiated directly.
3. All methods in an abstract class must be abstract.
4. Abstract classes cannot have constructors.

Question 4: Interfaces

What is the correct output of the following code?

```java
interface A {
    default void show() {
        System.out.println("Interface A");
    }
}

interface B {
    default void show() {
        System.out.println("Interface B");
    }
}

class C implements A, B {
    public void show() {
        A.super.show();
    }
}

public class Main {
    public static void main(String[] args) {
        C obj = new C();
        obj.show();
    }
}
```

1. `Interface A`
2. `Interface B`
3. Compilation Error
4. Runtime Exception

Question 5: Enums

Which of the following is true about enums in Java?

1. Enums cannot have methods.
2. Enum constants are implicitly static and final.
3. Enums can be extended by other classes.
4. Enum values cannot be compared using the `==` operator.

Question 6: Polymorphism

What will the following code output?

```java
class Parent {
    void display() {
        System.out.println("Parent");
    }
}

class Child extends Parent {
    void display() {
        System.out.println("Child");
    }
}

public class Main {
    public static void main(String[] args) {
        Parent obj = new Child();
        obj.display();
    }
}
```

1. `Parent`
2. `Child`
3. Compilation Error
4. Runtime Exception

Question 7: Access Modifiers

Which of the following statements is true about access modifiers in Java?

1. `protected` allows access only within the same package.
2. `private` members are accessible in subclasses.
3. `public` members are accessible from any other class.
4. `default` access level allows access from any package.

Question 8: `this` Keyword

What does the following code output?

```java
class Example {
    String name;

    Example(String name) {
        this.name = name;
    }

    void printName() {
        System.out.println(this.name);
    }
}

public class Main {
    public static void main(String[] args) {
        Example ex = new Example("Java");
        ex.printName();
    }
}
```

1. `null`
2. `Java`
3. Compilation Error
4. Runtime Exception

Question 9: Static Members

What is the output of the following code?

java Copy code

```java
class StaticExample {
    static int count = 0;

    StaticExample() {
        count++;
    }

    static int getCount() {
        return count;
    }
}

public class Main {
    public static void main(String[] args) {
        new StaticExample();
        new StaticExample();
        System.out.println(StaticExample.getCount());
    }
}
```

1. `1`
2. `2`
3. `0`
4. Compilation Error

Question 10: Overloading vs. Overriding

What is the key difference between method overloading and overriding in Java?

1. Overloading occurs at runtime, while overriding occurs at compile-time.
2. Overloading is when two methods in the same class have the same name but different parameters, while overriding is redefining a method in a subclass.
3. Overloading only applies to constructors, while overriding applies to any method.
4. Overloading requires the `@Override` annotation, while overriding does not.