Midterm

Sample Questions

Given the following class declarations and initializations in a client program, which of the following is a correct call to `method1`?

```
public class Test1
{
   public void method1(Test2 v1, Test3 v2)
   {
      // rest of method not shown
   }
}

public class Test2 extends Test1
{
}

public class Test3 extends Test2
{
}

The following initializations appear in a different class.
Test1 t1 = new Test1();
Test2 t2 = new Test2();
Test3 t3 = new Test3();
```

- A. t1.method1(t1,t1);
- B. t2.method1(t2,t2);
- C. t3.method1(t1,t1);
- D. t2.method1(t3,t2);
- E. t3.method1(t3,t3);

✔ Since `method1` is a public method of class `Test1` objects of any subclasses of `Test1` can invoke the method. So, it can be invoked on `t3` since it is an object of `Test3` and this is a subclass of `Test1`. And, since `method1` takes an object of class `Test2` and `Test3` as parameters. This actually means it can take an object of `Test2` or any subclass of `Test2` and an object of `Test3` or any subclass of `Test3`. So it can take `t3` which is an object of class `Test3` as an object of `Test2` since `Test3` is a subclass of `Test2`.

The concept of multiple inheritance is implemented in Java by

I. Extending two or more classes.

II. Extending one class and implementing one or more interfaces.

III. Implementing two or more interfaces.

- (II) and (III)

What will be the output?

```
interface A{
        public void method1();
}
class One implements A{
        public void method1(){
                System.out.println("Class One method1");
        }
}
class Two extends One{
        public void method1(){
                System.out.println("Class Two method1");
        }
}
public class Test extends Two{
        public static void main(String[] args){
                A a = new Two();
                a.method1();
        }
}
```

A. ○ Compilation Error

B. ○ Class One method1

C. ○ Class Two method1

D. ○ Throws a NoSuchMethodException at runtime.

E. ○ None of these

Determine output:

```
class A{
        public void method1(){
                System.out.print("Class A method1");
        }
}
class B extends A{
        public void method2(){
                System.out.print("Class B method2");
        }
}
class C extends B{
        public void method2(){
                System.out.print("Class C method2");
        }
        public void method3(){
                System.out.print("Class C method3");
        }
}
public class Test{
        public static void main(String args[]){
                A a = new A();
                C c = new C();
                c.method2();
                a = c;
                a.method3();
        }
}
```

A. ○ **Class B method2 Class C method3**

B. ○ **Class C method2 Class C method3**

C. ○ **Compilation Error**

D. ○ **Runtime exception**

E. ○ **None of these**

Answer: Option C

Solution:

It is important to understand that it is the type of reference variable - not the type of the object that it refers to - that which determines what members can be accessed. That is, when a reference to a subclass object is assigned to a super class reference variable, we will have access only to those parts of the object defined by the superclass.

In the above program method **method3()** is defined in the class C which is a subclass of B and so **A**. Even the reference variable a refers to c, a can't access **method3()** as this method is unknown to class **A**.

```java
public class Student {
    public String saySomething() {
        return "I love OOP";
    }
    public String say()  {
        return this.saySomething();
    }
}
```

```java
public class GradStudent extends Student{
    public String saySomething() {
        return "I love Advanced OOP";
    }
}
```

```java
public class StudTest {

    public static void main(String[] args) {
        Student s = new Student();
        GradStudent g = new GradStudent();
        Student gs = new GradStudent();
        Student sg = (Student) g;
        System.out.println(s.saySomething());
        System.out.println(g.saySomething());
        System.out.println(gs.saySomething());
        System.out.println(sg.saySomething());
    }
}
```

Output →

```
I love OOP
I love Advanced OOP
I love Advanced OOP
I love Advanced OOP
```

If we call say function for each object output →

```
I love OOP
I love Advanced OOP
I love Advanced OOP
I love Advanced OOP
```

17.

```
try{
      File f = new File("a.txt");
}catch(Exception e){
}catch(IOException io){
}
```

Is this code create new file name a.txt ?

A. ○ **true**

B. ○ **false**

C. ○ **Compilation Error**

D. ○ **None of these**

Answer: Option C

Solution:

IOException is unreachable to compiler because all exception is going to catch by Exception block.

20. Which of the below statement is/are true about Error?

A. An Error is a subclass of Throwable.

B. An Error is a subclass of Exception.

C. Error indicates serious problems that a reasonable application should not try to catch.

D. An Error is a subclass of IOException.

A. ○ A and D

B. ○ A and B

C. ○ A and C

D. ○ B and C

E. ○ B and D

Answer: Option C

Solution:

An Error is a subclass of Throwable that indicates serious problems that a reasonable application should not try to catch.

21.

```
class MyClass{
	MyClass(){
		System.out.print("one");
	}
	public void myMethod(){
		this();
		System.out.print("two");
	}
}

public class TestClass{
	public static void main(String args[]){
		MyClass obj = new MyClass();
		obj.myMethod();
	}
}
```

A. ○ two one one

B. ○ one one two

C. ○ one Exception

D. ○ Compilation Error

E. ○ None of these

Answer: Option D

Solution:

A method can't have constructor call.

Why is method overriding an essential part of polymorphism?

What is the purpose (why would you do it) of making a class abstract?

Explain the difference between static (early) and dynamic(late) binding.

And some True/False Questions