Java Fundamentals 7-5:

Polymorphism

Practice Activities

Try It/Solve It:

1. What will be the output of the following code?

```
1  package nnn;
2
3  class A {
4      void callthis() {
5          System.out.println("Inside Class A's Method!");
6      }
7  }
8
9  class B extends A {
10     void callthis() {
11         System.out.println("Inside Class B's Method!");
12     }
13 }
14
15 class C extends A {
16     void callthis() {
17         System.out.println("Inside Class C's Method!");
18     }
19 }
20
21 public class DynamicDispatch {
22     public static void main(String[] args) {
23         A a = new A();
24         B b = new B();
25         C c = new C();
26
27         A ref;
28
29         ref = b;
30         ref.callthis();
31         ref = c;
32         ref.callthis();
33         ref = a;
34         ref.callthis();
35     }
36 }
37
```

Problems @ Javadoc Declaration Console ×

<terminated> DynamicDispatch [Java Application] C:\Users\e020ax\.p2\poo

```
Inside Class B's Method!
Inside Class C's Method!
Inside Class A's Method!
```

2. What is the difference between an abstract class and an interface? When is it appropriate to use an abstract class or an interface?

Key Differences

- **Abstract Class:**
    - Can provide a partial implementation.
    - Can hold state (instance variables).
    - Supports constructor.
    - A class can only extend one abstract class.
- **Interface:**
    - Cannot provide an implementation (except for default and static methods).
    - Cannot hold state (fields are constants).
    - Does not support constructors.
    - A class can implement multiple interfaces.

Choosing Between Abstract Class and Interface

- Use an abstract class when you need to share code among closely related classes and also need to enforce some methods to be implemented by subclasses.
- Use an interface when you want to define a contract that can be implemented by classes from different class hierarchies, or when you need to support multiple inheritance of type.

3.Given the information for the following, determine whether they will result: Always compile, sometimes compile, or does not compile. public interface A public class B implements A public abstract class C public class D extends C public class E extends B Each class have been initialized, but it is not clear what they have been initialized to: A a = new... B b = new... C c = new... D d = new... E e = new... The following methods are included: interface A specifies method void methodA() class C has the abstract method void methodC()

```
public interface A {
    void methodA();
}

public class B implements A {
    @Override
    public void methodA() {
        // Implementation of methodA
    }
}

public abstract class C {
```

```
    abstract void methodC();
}

public class D extends C {
    @Override
    void methodC() {
        // Implementation of methodC
    }
}

public class E extends B {
    // No additional methods or fields
}
```

code	Always Compile, Sometimes Compile, or Does Not Compile?
a = new B();	Always Compile
d = new C();	Does Not Compile
b.methodA();	Always Compile
e.methodA();	Always Compile
c = new C();	Does Not Compile
(D)c.methodC();	Does Not Compile

4. Override the toString() method for the class below to output the results, matching the given output. The toString() method should print all the values from 1 to the number specified in num and then print the final value using the provided getFactorial method. Assume the variable int num is a public global value: “Factorial: 10! = 1 * 2 * 3 * 4 * 5 * 6 * 7 * 8 * 9 * 10 = 3628800”

```java
package nnn;

public class FactorialCalculator {
    public int num;

    public int getFactorial() {
        int factorial = 1;
        for (int i = num; i > 0; i--) {
            factorial *= i;
        }
        return factorial;
    }

    public String toString() {
        StringBuilder result = new StringBuilder();
        result.append("Factorial: ").append(num).append("! = ");

        for (int i = 1; i <= num; i++) {
            if (i > 1) {
                result.append(" * ");
            }
            result.append(i);
        }

        result.append(" = ").append(getFactorial());

        return result.toString();
    }

    public static void main(String[] args) {
        FactorialCalculator calc = new FactorialCalculator();
        calc.num = 10;
        System.out.println(calc.toString());
    }
}
```

Problems @ Javadoc Declaration Console ×

<terminated> FactorialCalculator [Java Application] C:\Users\e020ax\.p2\p

```
Factorial: 10! = 1 * 2 * 3 * 4 * 5 * 6 * 7 * 8 * 9 * 10 = 3628800
```