Menu

15 Top Java Method Overriding Interview Programs for Practice

In this tutorial, we have listed the most important top 15 Java method overriding interview programs for practice.

Here, you can solve the interview coding questions based on concepts of method overriding, covariant return type, exceptional handling with method overriding, the effect of access modifiers on method overriding.

All the interview coding programming questions are the most important for Java beginners and 2-3 years of experience.

If you are able to find out the output of more than 10 coding questions, you can definitely solve any questions related to method overriding in any Java technical tests or interviews.

1. Is the below code successfully overridden?

```
package overridingPrograms;
public class A
{
private void m1()
{
 System.out.println("m1-A");
 }
}
public class B extends A
```

```
 System.out.println("m1-B");
}
public static void main(String[] args)
{
 B b = new B();
 b.m1();
 }
}
```

Ans: No, Compile time error.

Explanation: The overriding concept is not applicable to the private method. Parent class private method is not visible in the child class. Keep in mind. Based on our requirement, we can define exactly the same private method in the child class. The code is valid but not overriding.

2. Will the below code compile successfully?

```
package overridingPrograms;
public class A
{
public final void m1()
{
 System.out.println("m1-A");
 }
}
public class B extends A
{
public void m1()
{
 System.out.println("m1-B");
}
public static void main(String[] args)
{
 B b = new B();
  b.m1();
 }
```

Ans:No, Compile-time error.

Explanation: If the parent class method is declared as final, you cannot override it in the child class. If you are trying to override it, you will get the compile-time error-"Overridden method is final".

3. Will the below code compile successfully? If yes, what will be the output?

```
package overridingPrograms;
public class A
{
protected void m1()
{
 System.out.println("m1-A");
 }
}
public class B extends A
{
public final void m1()
{
 System.out.println("m1-B");
}
public static void main(String[] args)
{
 B b = new B();
  b.m1();
```

```
 }
}
```

Ans: Yes, the code will be successfully compiled.

Explanation: You cannot declare the parent class method as final but the child class method can be declared as final. It is used to restrict further overriding.

a) When b.m1() will be executed, it will call m1() of class B because the reference variable 'b' is pointing to the objects of class B. So, the output will be "m1-B".

b) Since 'a' is also pointing to the objects of class B. Therefore, the output will be "m1-B".

You can get more detailed reasons in this tutorial: *Method overriding in Java*

4. What will be the output of the following program?

```
package overridingPrograms;
public class A
{
 synchronized void m1()
 {
  System.out.println("m1-A");
 }
void m2(char c)
{
 System.out.println("m2-A");
 }
```

```
{
@Override
public final void m1()
{
 System.out.println("m1-B");
}
@Override
synchronized void m2(char c)
{
 System.out.println("m2-B");
}
public static void main(String[] args)
{
 A a = new B();
 a.m1();
 a.m2('a');
 }
}
```

```
Output:
      m1-B
      m2-B
```

span style=”color: #ff0000;”>**Explanation:** When an overridden method is synchronized, the overriding method can be non-synchronized and vice-versa.

```
package overridingPrograms;
public class X
{
strictfp void method(int a)
{
 System.out.println("One");
}
strictfp void method(double b)
{
 System.out.println("Two");
 }
}
public class Y extends X
{
@Override
void method(double b)
{
 System.out.println("Three");
 }
}
public class Test
{
public static void main(String[] args)
{
  new Y().method(20);
 }
}
```

```
Output:
      One
```

Explanation: If the parent class method is strictfp, the child class method can be non-strictfp and vice-versa.

6. What will be the output of the following program?

```
package overridingPrograms;
public class X
{
void draw(int a, float b) throws Throwable
{
 System.out.println("Circle");
 }
}
public class Y extends X
{
@Override
void draw(int a, float b)
{
 System.out.println("Rectangle");
 }
}
public class Z extends Y
{
@Override
void draw(int a, float b) throws ArithmeticException
{
 System.out.println("Square");
 }
}
public class Test
{
public static void main(String[] args) throws Throwable
```

```
 x.draw(20, 30.5f);
 Y y = (Y)x;
 y.draw(10,2.9f);
 Z z = (Z)y;
 z.draw(20, 30f);
 }
}
```

```
Output:
    Rectangle
    Rectangle
    ClassCastException
```

Explanation:

1. x.draw(20,30.5f); will call draw method of class Y because the reference variable is pointing to the objects of class Y. Therefore, the output is “Rectangle”.

2. When the statement Y y = (Y)x; will be executed, JVM will perform downcasting. When the reference variable of child class refers to the object of a parent class, it is known as downcasting.

Here, the reference variable y is pointing to the reference x of the parent class X but x is pointing to the objects of Class Y. So, y is pointing to the objects of class Y. Therefore, y.draw(10, 2.9f); will call draw method of class Y and the output will be “Rectangle”.

3. When the statement Z z = (Z)y; will be executed, JVM will throw ClassCastException because Y cannot be cast to Z.

7. What will be the output of the following program?

```
package overridingPrograms;
public class Animal
{
void m1(Animal a)
{
 System.out.println("Both dogs and cats are pets.");
```

```
{
@Override
protected void m1(Animal a)
{
 System.out.println("The only dog is a pet animal.");
 }
}
public class Cat extends Dog
{
@Override
public void m1(Animal a)
{
 System.out.println("Cat is also a pet animal.");
 }
}
public class AnimalClass
{
public static void main(String[] args)
{
 Animal a = new Cat();
  a.m1(null);
 Dog d = new Dog();
  d.m1(a);
 Cat c = (Cat)new Animal();
  c.m1(null);
 }
}
```

Output:

Cat is also a pet animal.
The only dog is a pet animal.
Exception in thread "main" java.lang.ClassCastException.

8. Can you find out the error in the below code?

```
{
 System.out.println("Class P");
 }
}
public class Q extends P
{
@Override
static void m1()
{
 System.out.println("Class Q");
 }
}
```

Ans: The overriding concept is not applicable to a static method.

9. What will be the output of the following program?

```
public class P
{
void m1(Number n)
{
 System.out.println("m1-P");
 }
}
public class Q extends P
{
void m1(double d)
{
 System.out.println("m2-Q");
}
public static void main(String[] args)
{
 Q q = new Q();
  q.m1(1);
 q.m1(null);
 }
}
```

```
Output:
        m2-Q
        m1-P
```

10. In the below example program, class Q extends class P. Which method of class Q is not properly overridden in class B?

```
public class P
{
public Object m1()
{
 return null;
}
void m2(Number n)
{
 System.out.println("m1-P");
 }
}
public class Q extends P
{
public StringBuffer m1()
{
 return null;
}
void m2(double d)
{
 System.out.println("m2-Q");
 }
}
```

Ans:m2() is not properly overridden in class Q because method signature must be the same while overriding.

11. Can you identify error in the below code snippet?

```
{
 return null;
 }
}
public class Q extends P
{
public String m1(char c) throws IOException
{
 return null;
 }
}
```

Ans: If an overridden method does not throw an exception then the overriding method cannot throw any checked or compile-time exception. IOException is a compile-time exception.

For more details, go to this link: Rules of Exceptional handling with method overriding

12. What will be the output of the following program?

```
public class XY
{
protected Number m1(int a)
{
 System.out.println("One");
  return null;
}
Object m2()
{
 System.out.println("Two");
  return null;
 }
}
public class YZ extends XY
{
protected String m2()
{
```

```
}
public class XYZ
{
public static void main(String[] args)
{
 XY xy = new YZ();
  xy.m1(20);
  xy.m2();
 }
}
```

```
Output:
      One
      Three
```

For explanation, go to this link: Covariant return type in Java

13. In the below snippet code, Is m1() correctly overridden in the subclasses of class X?

```
public class One
{
void m1()
{

 }
}
public class Two extends One
{
@Override
protected void m1()
{
 System.out.println("m1-Two");
 }
}
```

```
public void m1()
{
 System.out.println("m1-Three");
 }
}
```

Ans: Yes

Explanation: m1() method is correctly overridden in the subclasses of class X.

14. What will be the output of the following program?

```
package overridingPrograms;
public class One
{
void m1() throws Throwable
{
 System.out.println("m1-One");
 }
}
public class Two extends One
{
@Override
protected void m1() throws Exception
{
 System.out.println("m1-Two");
 }
}
import java.io.IOException;
public class Three extends Two
{
@Override
public final void m1() throws IOException
{
 System.out.println("m1-Three");
 }
}
```

```
{
 One o = new Two();
  o.m1();
 Two t = new Three();
  t.m1();
 Three th = new Three();
  th.m1();
 }
}
```

```
Output:
      m1-Two
      m1-Three
      m1-Three
```

15. Can you identify how many errors are in the below snippet code?

```
public class Rose
{
protected void color(char r)
{
 System.out.println("Red");
 }
}
public class Marigold extends Rose
{
@Override
void color(char m) throws NullPointerException
{
 System.out.println("Orange");
 }
}
public class Sunflower extends Marigold
{
@Override
```

```
    }
}
```

```
Output:
    Two
```

Explanation:

1. You cannot reduce the visibility of the overriding method in subclass Marigold.
2. You cannot declare the checked exception while overriding in the subclass Sunflower. Marigold is the superclass of Sunflower.

Recommended Tutorials:

- Java Method Overloading Interview Coding Questions
- Top 10 Inheritance Interview Coding Questions for Practice

Hope that you will have practiced the top 15 method overriding interview programs that are based on method overriding. All coding questions are very important for Java interviews and technical test purposes.
Thanks for reading!!!

Java Interview Programs

Leave a Comment

You must be logged in to post a comment.

Java Tutorials

- Java Introduction +
- Basics of Java +
- Java Class and Object +
- Java Data types and Variables +
- Java Operators +
- Decision Making, Branching, Looping +
- Java Packages +
- Java Methods +
- Java Constructor +
- Java Modifiers +
- Blocks in Java +
- Java Static and Final Keywords +
- Inner Classes in Java +
- OOPs Concepts in Java +
- Java Encapsulation +
- Inheritance in Java +
- Java Super and This keywords +

Java Abstraction +

Java Polymorphism +