



Menu

Java Method Overloading Interview Programs for Practice

Here, we have listed a sample of top **8 Java method overloading interview programs for practice.**



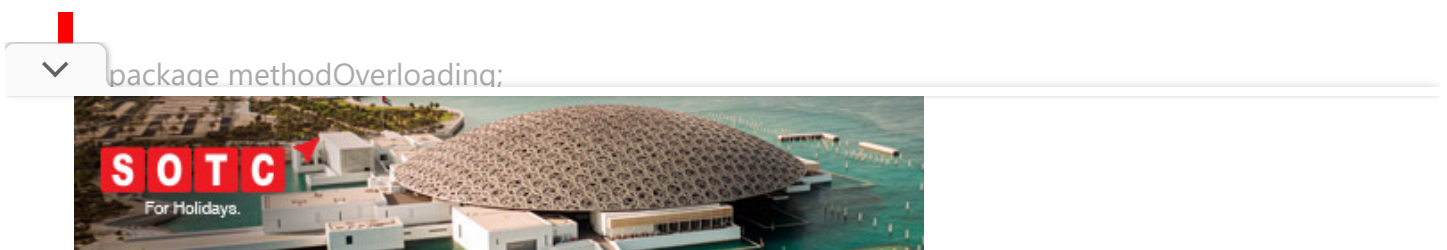
All these questions are very important that can be asked in technical tests or interviews for freshers and 1 to 3 years of experience.

If you try to solve all these questions based on method overloading concept, your concept will be more clear and could be able to solve questions easily in the technical tests.

We have also explained the explanation of difficult question programs so that you do not get any problems in understanding **method overloading concepts**. So, let's try.

Java Interview Questions on Method Overloading

Question 1:



```
}  
public class B extends A  
{  
  
}  
public class C extends B  
{  
  
}
```

Consider the above program source code. What will be the output of the following scenarios?

Scenario 1:

```
public class OverLoadingScenarios  
{  
void m1(A a) {  
    System.out.println(" I am in m1-A");  
}  
void m1(B b) {  
    System.out.println("I am in m1-B");  
}  
void m1(C c) {  
    System.out.println("I am in m1-C");  
}  
}  
public class OverLoadingTest {  
public static void main(String[] args)  
{  
    OverLoadingScenarios obj = new OverLoadingScenarios();  
    // Scene 1:  
    A a = new A();  
    obj.m1(a);  
    // Scene 2:  
    B b = new B();  
    obj.m1(b);  
}
```



```
// Scene 4:
    B bc = new C();
    obj.m1(bc);
// Scene 5:
    A ab = new B();
    obj.m1(ab);
}
}
```



Output:

Scene 1:

I am in m1-A

Scene 2:

I am in m1-B

Scene 3:

I am in m1-C

Scene 4:

I am in m1-B

Scene 5:

I am in m1-A

Explanation:

1. Scene 1: 'a' is the reference variable of class A and pointing to object of class A. In resolving method overloading, during compilation, Java compiler does not check types

of object to which particular reference variable is pointing.



In scene 1, 'a' is the reference variable of class A. Therefore, the compiler will call m1() method of A because it is exactly matched. Similarly, for scenes 2, and 3.

2. Scene 4: 'bc' is the reference variable of class B and it is pointing to object of class C but Java compiler will call m1() method of class B because bc is the reference variable of class B. Similarly for scene 5.

Key points:

In method overloading, method resolution takes place entirely at compile-time and It is always resolved by the compiler based on reference type only. In overloading, runtime object does not play any role.

Scenario 2:

```
public class OverLoadingScenarios2 {  
    void m1(A a)  
    {  
        System.out.println(" I am in m1-A");  
    }  
    void m1(B b) {  
        System.out.println("I am in m1-B");  
    }  
    void m1(C c) {  
        System.out.println("I am in m1-C");  
    }  
    public static void main(String[] args)  
    {  
        OverLoadingScenarios2 obj = new OverLoadingScenarios2();  
        obj.m1(null);  
    }  
}
```

Output:

I am in m1-C

✓ Explanation:



2. A call to m1() method with null argument executes the third version of m1() method with argument c of type C because while resolving overloaded method, the compiler always uses the presidency for child type argument. In this kind of case, you always see the parent-child relationship.



Mock Test for GATE C: Exam

If you aspiring GATE 2023, then this right place for you to be in

GeeksforGeeks

For practice, more questions, go to this link: [Automatic type promotion in method overloading Java](#)

Note: A field of an object type can have a null value in Java. The default value for string and an object type argument is null.

Scenario 3:

```
public class OverLoadingScenarios3
{
    void m1(A a) {
        System.out.println(" I am in m1-A");
    }
    void m1(B b) {
        System.out.println("I am in m1-B");
    }
    void m1(Object o) {
        System.out.println("I am in m1-C");
    }
    public static void main(String[] args)
    {
        OverLoadingScenarios3 obj = new OverLoadingScenarios3();
        obj.m1(null);
    }
}
```



Output:

I am in m1-B

Explanation:

An object is the superclass of A and B where A is the superclass of B. Thus, B is the child class. Therefore, the output is "I am in m1 B".

Scenario 4:

```
public class OverLoadingScenarios4
{
    void m1(A a) {
        System.out.println(" I am in m1-A");
    }
    void m1(String s) {
        System.out.println("I am in m1-B");
    }
    void m1(Object o) {
        System.out.println("I am in m1-C");
    }
    public static void main(String[] args)
    {
        OverLoadingScenarios2 obj = new OverLoadingScenarios2();
        obj.m1(null);
    }
}
```

Output:

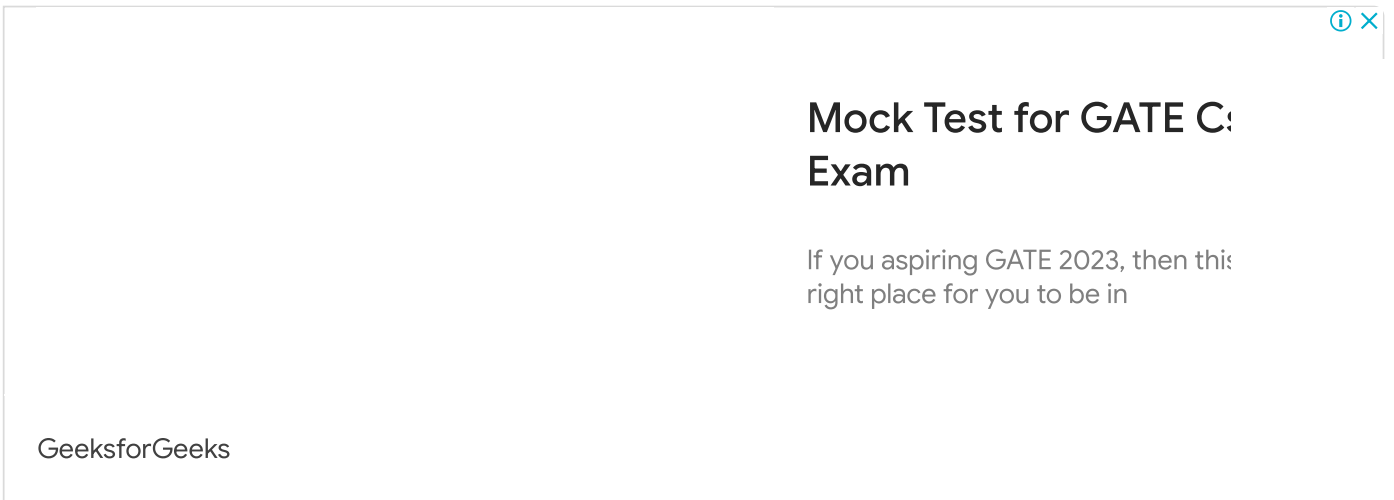
Unresolved compilation problem: The method m1(A) is ambiguous for the type OverLoadingScenarios2

Explanation:

✓ In this scenario, Object class is superclass of string class and class A at the same level.



When we will pass null argument for calling m1() method, we have created an ambiguous situation for the compiler that cannot determine which method to call because all three methods are exact matches for our call.



Mock Test for GATE C++ Exam

If you aspiring GATE 2023, then this is the right place for you to be in

GeeksforGeeks

Therefore, the compiler complains of an error that the call is **ambiguous** and it will generate compile-time error.

Note: An overloaded method cannot be ambiguous its own. It only becomes ambiguous if you create an ambiguous situation.

Question 2: What will be the output of the following program?

```
package methodOverloading;
public class XYZ
{
    void msg(Object obj) {
        System.out.println("Good");
    }
    void msg(String str) {
        System.out.println("Better");
    }
    void msg(Integer itr) {
        System.out.println("Best");
    }
    public static void main(String[] args)
    {
        XYZ obj = new XYZ();
        obj.msg(new Object()); // Exact matched to call m1() method with Object type argument.
        obj.msg("Sciencetech Easy"); // Exact matched to call m1() method with String type argument.
```



```
obj.msg(new String());  
obj.msg(10); // Exact matched to call m1() with Integer type argument.  
obj.msg(new Integer(0));  
}  
}
```

Output:

Good
Better
Good
Better
Best
Best

Question 3: What will be the output of the below program?

```
package methodOverloading;  
public class XYZ  
{  
    void msg(Object obj) {  
        System.out.println("Good");  
    }  
    void msg(String str) {  
        System.out.println("Better");  
    }  
    void msg(Integer itr) {  
        System.out.println("Best");  
    }  
    public static void main(String[] args)  
    {  
        XYZ obj = new XYZ();  
        obj.msg(null); // Created an ambiguous situation for the compiler to call  
    }  
}
```



Output:

Compile time error: The method msg(Object) is ambiguous for the type XYZ

Question 4: What is the output of the below program?

```
package methodOverloading;
public class Overloaded
{
    public static void m1(int a) {
        System.out.println("int");
    }
    public static void m1(short a) {
        System.out.println("short");
    }
    public static void m1(Object a) {
        System.out.println("object");
    }
    public static void m1(String a)
    {
        System.out.println("String");
    }
    public static void main(String[] args)
    {
        byte b = 5;
        m1(b); // First call
        m1(5); // Second call Integer
    }
}
```



```
m1(null); // Fifth call  
}  
}
```

Output:

```
short  
int  
object  
String  
String
```

Explanation:

1. When we will call `m1()` method with byte argument, there is no `m1()` definition that will take byte as an argument.

Since the smaller data type in size is short than int data type which is larger size as a comparison to short. Therefore, the compiler will promote byte to short. The result will be short.

2. In the second method call, an exact match is found to call `m1()` with int argument.

3. The third method call is `m1(i)` where i is a type of Integer. There is no such `m1()` method that takes an integer type argument.

Therefore, the compiler will implicitly upcast from Integer to Object type argument and it will call to `m1()` method with Object type argument.

Remember that the compiler allows implicit upcast, not downcast. Therefore, `m1(int a)` will not be considered.

4. In the fourth method call, an exact match is found to call `m1()` with String argument.

5. The last method call is `m1(null)`. Since null is valid for both object and string. So, which method will be called? `m1(String a)` will be called because string is child class of object class.

Question 5: What is the output of the below program source code?



```
public static void msg(long a, int b) {  
    System.out.println("Hello");  
}  
public static void msg(int a, long b) {  
    System.out.println("Hi");  
}  
public static void main(String[] args) {  
    msg(5l, 10);  
    msg(10,11);  
}  
}
```

Output:

Hello

Unresolved compilation problem: The method msg(long, int) is ambiguous for the type Overloaded

Question 6: What will be the outcome of the below program?

```
package methodOverloading;  
public class Overloaded  
{  
    public static void test(int[] intArr) {  
        System.out.println("int array");  
    }  
    public static void test(char[] charArr) {  
        System.out.println("char array");  
    }  
    public static void main(String[] args)  
    {  
        test(null);  
    }  
}
```



Explanation:

int[] and char[] both are not primitive type in Java. Both are classes that extend object class at the same level. Therefore, the compiler is unable to resolve overloaded method to call.

Question 7: What will be the outcome of the below program?

```
package methodOverloading;
public class Overloaded
{
    public void test(int i) {
        System.out.println("int");
    }
    public void test(Number n) {
        System.out.println("Number");
    }
    public void test(Integer i) {
        System.out.println("Integer");
    }
    public static void main(String[] args)
    {
        Overloaded o = new Overloaded();
        o.test(null);
        o.test(10); // Exact matched.
    }
}
```

Output:

Integer
int

Explanation:

✓ Object class is the superclass of Number whereas Number is parent class of Integer.



```
package methodOverloading;

public class Overloaded
{
    public void test(int i) {
        System.out.println("Int");
    }

    public void test(int... i) {
        System.out.println("Int");
    }

    public void test(char... c) {
        System.out.println("Char varargs");
    }

    public static void main(String[] args)
    {
        Overloaded obj = new Overloaded();
        obj.test('a');
        obj.test(10); // Exact matched.
    }
}
```

Output:

Int

Int

Explanation:

Methods with varargs (...) have the lowest priority. That's why the output is Int.

Recommended Post:

- [Top 15 Java Method Overriding Interview Programs for Practice](#)
- [Top 32+ OOPs Interview Questions](#)
- [Top 10 Java Inheritance Interview Programs for Practice](#)
- [50 Java Interface Interview Programming Questions](#)



Hope that this tutorial has covered almost all the variety of Java method overloading interview programs that you will have practiced and enjoyed them.

Thanks for reading!!!

- Java Programs
- Java Method Overloading Coding Questions for Interview

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	+



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 Overloading	+
Java Overriding	+
Java Abstraction	+
Java Polymorphism	+

[Home](#) [About Us](#) [Contact](#) [Privacy Policy](#)

© Copyright 2018-2022 Sciencetech Easy. All rights reserved.

