
Question 1

What will happen if you compile/run this code?

```
1: public class Q1 extends Thread
2: {
3:     public void run()
4:     {
5:         System.out.println("Before start method");
6:         this.stop();
7:         System.out.println("After stop method");
8:     }
9:
10: public static void main(String[] args)
11: {
12:     Q1 a = new Q1();
13:     a.start();
14: }
15: }
```

- A) Compilation error at line 7.
- B) Runtime exception at line 7.
- C) Prints "Before start method" and "After stop method".
- D) Prints "Before start method" only.**

Question 2

What will happen if you compile/run the following code?

```
1: class Test
2: {
3:     static void show()
4:     {
5:         System.out.println("Show method in Test class");
6:     }
7: }
8:
9: public class Q2 extends Test
10: {
11:     static void show()
12:     {
13:         System.out.println("Show method in Q2 class");
14:     }
15:     public static void main(String[] args)
16:     {
17:         Test t = new Test();
18:         t.show();
19:         Q2 q = new Q2();
20:         q.show();
21:
22:         t = q;
23:         t.show();

```

```
24:  
25:     q = t;           incompatable types  
26:     q.show();  
27: }  
28: }
```

A) prints "Show method in Test class"
"Show method in Q2 class"
"Show method in Q2 class"
"Show method in Q2 class"

B) prints "Show method in Test class"
"Show method in Q2 class"
"Show method in Test class"
"Show method in Test class"

C) prints "Show method in Test class"
"Show method in Q2 class"
"Show method in Test class"
"Show method in Q2 class"

D) Compilation error.

Question 3

The following code will give

```
1: class Test  
2: {  
3:     void show()  
4:     {  
5:         System.out.println("non-static method in Test");  
6:     }  
7: }  
8: public class Q3 extends Test  
9: {  
10:     static void show()  
11:     {  
12:         System.out.println("Overridden non-static method in Q3");  
13:     }  
14:  
15:     public static void main(String[] args)  
16:     {  
17:         Q3 a = new Q3();  
18:     }  
19: }
```

show() in Q3 cannot override show() in Test
for overriding methods function signature must be same

- A) Compilation error at line 3.
- B) Compilation error at line 10.**
- C) No compilation error, but runtime exception at line 3.

D) No compilation error, but runtime exception at line 10.

Question 4

The following code will give

```
1: class Test
2: {
3:     static void show()
4:     {
5:         System.out.println("Static method in Test");
6:     }
7: }
8: public class Q4 extends Test
9: {
10:     static void show()
11:     {
12:         System.out.println("Overridden static method in Q4");
13:     }
14:     public static void main(String[] args)
15:     {
16:     }
17: }
```

show() in Q4 cannot override show() in Test
for overriding methods function signature must be same
No error if this ("static") keyword added only exception is you cannot use "super" and "this"
keywords in static methods

- A) Compilation error at line 3.
- B) Compilation error at line 10.**
- C) No compilation error, but runtime exception at line 3.
- D) No compilation error, but runtime exception at line 10.

Question 5

The following code will print

```
1: int i = 1;
2: i <<= 31;
3: i >>= 31;
4: i >>= 1;
5:
6: int j = 1;
7: j >>= 31;
8: j >>= 31;
9:
10: System.out.println("i = " + i);
11: System.out.println("j = " + j);
```

- A) i = 1
j = 1

B) i = -1
j = 1

C) i = 1
j = -1

D) i = -1
j = 0

Question 6

The following code will print

```
1: Double a = new Double(Double.NaN);
2: Double b = new Double(Double.NaN);
3:
4: if( Double.NaN == Double.NaN )
5:     System.out.println("True");
6: else
7:     System.out.println("False");
8:
9: if( a.equals(b) )
10:    System.out.println("True");
11: else
12:    System.out.println("False");
```

A) True
True

B) True
False

C) False
True

D) False
False

Question 7

The following code will print

```
1: if( new Boolean("true") == new Boolean("true"))
2:     System.out.println("True");
3: else
4:     System.out.println("False");
```

A) Compilation error.

B) No compilation error, but runtime exception.

C) Prints "True".

D) Prints "False".

Question 8

The following code will give

```
1: public class Q8
2: {
3:     int i = 20;
4:     static
5:     {
6:         int i = 10; //local to this block
7:
8:     }
9:     public static void main(String[] args)
10:    {
11:        Q8 a = new Q8();
12:        System.out.println(a.i);
13:    }
14: }
```

- A) Compilation error, variable "i" declared twice.
- B) Compilation error, static initializers for initialization purpose only.
- C) Prints 10.
- D) Prints 20.**

Question 9

The following code will give

```
1: Byte b1 = new Byte("127");
2:
3: if(b1.toString() == b1.toString())
4:     System.out.println("True");
5: else
6:     System.out.println("False");
```

- A) Compilation error, toString() is not available for Byte.
- B) Prints "True".
- C) Prints "False".**

Question 10

What will happen if you compile/run this code?

```
1: public class Q10
2: {
3:     public static void main(String[] args)
4:     {
5:         int i = 10;
6:         int j = 10;
```

```
7:    boolean b = false;
8:
9:    if( b = i == j)
10:       System.out.println("True");
11:    else
12:       System.out.println("False");
13:   }
14: }
```

- A) Compilation error at line 9 .
- B) Runtime error exception at line 9.
- C) Prints "True".
- D) Prints "False".

Question 11

What will happen if you compile/run the following code?

```
1:  public class Q11
2:  {
3:      static String str1 = "main method with String[] args";
4:      static String str2 = "main method with int[] args";
5:
6:      public static void main(String[] args)
7:      {
8:          System.out.println(str1);
9:      }
10:
11:     public static void main(int[] args)
12:     {
13:         System.out.println(str2);
14:     }
15: }
```

- A) Duplicate method main(), compilation error at line 6.
- B) Duplicate method main(), compilation error at line 11.
- C) Prints "main method with main String[] args".
- D) Prints "main method with main int[] args".

Question 12

What is the output of the following code?

```
1:  class Test
2:  {
3:      Test(int i)
4:      {
5:          System.out.println("Test(" +i +"");
6:      }
7:  }
8:
9:  public class Q12
```

```
10: {
11:     static Test t1 = new Test(1);
12:
13:     Test    t2 = new Test(2);
14:
15:     static Test t3 = new Test(3);
16:
17:     public static void main(String[] args)
18:     {
19:         Q12 Q = new Q12();
20:     }
21: }
```

Static variable are initialized when the class is loaded to JVM but Instance variable are initialized when the object is created.

A) Test(1)
Test(2)
Test(3)

B) Test(3)
Test(2)
Test(1)

C) Test(2)
Test(1)
Test(3)

D) Test(1)
Test(3)
Test(2)

Question 13

What is the output of the following code?

```
1: int i = 16;
2: int j = 17;
3:
4: System.out.println("i >> 1 = " + (i >> 1));
5: System.out.println("j >> 1 = " + (j >> 1));
```

A) Prints i >> 1 = 8
j >> 1 = 8

B) Prints i >> 1 = 7
j >> 1 = 7

C) Prints i >> 1 = 8
j >> 1 = 9

D) Prints i >> 1 = 7
j >> 1 = 8

Question 14

What is the output of the following code?

```
1: int i = 45678;
2: int j = ~i;
3:
4: System.out.println(j);
```

- A) Compilation error at line 2. ~ operator applicable to boolean values only.
- B) Prints 45677.
- C) Prints -45677.
- D) Prints -45679.**

Question 15

What will happen when you invoke the following method?

```
1: void infiniteLoop()
2: {
3:     byte b = 1;
4:
5:     while ( ++b > 0 )
6:         ;
7:     System.out.println("Welcome to Java");
8: }
```

- A) The loop never ends(infiniteLoop).
- B) Prints "Welcome to Java".**
- C) Compilation error at line 5. ++ operator should not be used for byte type variables.