OCA — 1Z0-808

Oracle Certified Associate, Java SE 8 Programmer I

Are You ready for OCA Exam?

Do you want to check your preparation?

Then you have come to the right place

```
public static void go()
{       int x = 3;
        int y = 2;
        System.out.println("In method go. x: " + x + " y: " + y);
        falseSwap(x,y);
        System.out.println("in method go. x: " + x + " y: " + y);
        moreParameters(x,y);
        System.out.println("in method go. x: " + x + " y: " +
}

public static void falseSwap(int x, int y)
{       System.out.println("in method falseSwap. x: " + x + "
        int temp = x;
        x = y;
        y = temp;
        System.out.println("in method falseSwap. x: " + x + " y: " +
}

public static void moreParameters(int a, int b)
{       System.out.println("in method moreParameters. a: " + a +
```

```
public static void go()
{       int x = 3;
        int y = 2;
        System.out.println("In method go. x: "
        falseSwap(x,y);
        System.out.println("in method go. x: "
        moreParameters(x,y);
        System.out.println("in method go. x: "
}

public static void falseSwap(int x, int y)
{       System.out.println("in method falseSwap. x: " +
        int temp = x;
        x = y;
        y = temp;
        System.out.println("in method falseSw      + x + " y: " + y);
}

public static void moreParameters(int a, int
{       System.out.println("in method morePa   meters. a: " + a + " b: " + b);
        a = a * b;
        b = 12;
        System.out.println("in method morePa  ameters. a: " + a + " b: " + b);
        falseSwap(b,a);
        System.out.println("in method morePa  ameters. a: " + a + " b: " + b);
```

OCA — 1Z0-808

Oracle Certified Associate, Java SE 8 Programmer I

Never Seen before Questions

25 Practice Test Questions with Explanation

Extracted from Best Selling Course on Udemy

OCA — 1Z0-808
Exam Practice Test

Hello Everyone
My Name is Udayan Khattry
This slideshow is an excerpt from my popular course OCA-1Z0-808 Practice Test on Udemy which has helped 100s of students to pass the exam.
Every single question is prepared according to EXAM curriculum and objectives.
Detailed Explanation is also provided along with the answer

Hope you enjoy these questions..

1

Consider 3 files given below:

```
 //Order.java
package orders;

public class Order {

}

//Item.java
package orders.items;

public class Item {

}

//Shop.java
package shopping;

/*INSERT*/

public class Shop {
    Order order = null;
    Item item = null;
}
```

For the class Shop, which options, if used to replace /*INSERT*/, will resolve all the compilation errors? Select 2 options.

Options

```
A. import orders.Order;
    import orders.items.Item;
B. import orders.*;
C. Import orders.items.*;
D. import orders.*;
    import orders.items.*;
E. import orders.*;
    import items.*;
```

Answer

[✓ A. `import orders.Order;`
`import orders.items.Item;`]

B. `import orders.*;`

C. `Import orders.items.*;`

[✓ D. `import orders.*;`
`import orders.items.*;`]

E. `import orders.*;`
`import items.*;`

Answer

```
✓A. import orders.Order;
     import orders.items.Item;
 B. import orders.*;
```

If you check the directory structure, you will find that directory "orders" contains "items", but orders and orders.items are different packages.
import orders.*; will only import all the classes in orders package but not in orders.items package.

You need to import Order and Item classes.

To import Order class, use either import orders.Order; OR import orders.*; and to import Item class, use either import orders.items.Item; OR import orders.items.*;

2

What will be the result of compiling and executing Test class?

```
import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<>();
        list.add(100);
        list.add(200);
        list.add(100);
        list.add(200);
        list.remove(100);

        System.out.println(list);
    }
}
```

Options

A. [200, 100, 200]

B. [100, 200, 200]

C. [200, 200]

D. [200]

E. Compilation error

F. Exception is thrown at runtime

Answer

A. [200, 100, 200]
B. [100, 200, 200]
C. [200, 200]
D. [200]
E. Compilation error
[✓ F. Exception is thrown at runtime]

Answer

A. [200, 100, 200]

B. [100, 200, 200]

List cannot accept primitives, it can accept objects only. So, when 100 and 200 are added to the list, then auto-boxing feature converts these to wrapper objects of Integer type.
So, 4 items gets added to the list. One can expect the same behavior with remove method as well that 100 will be auto-boxed to Integer object.
But remove method is overloaded in List interface: remove(int) => Removes the element from the specified position in this list and remove(Object) => Removes the first occurrence of the specified element from the list.
As remove(int) version is available, which perfectly matches with the call remove(100); hence compiler does not do auto-boxing in this case.
But at runtime remove(100) tries to remove the element at 100th index and this throws IndexOutOfBoundsException.

3

For the class Test, which options, if used to replace /*INSERT*/, will print "Lucky no. 7" on to the console? Select 3 options.

```
public class Test {
    public static void main(String[] args) {
        /*INSERT*/
        switch(var) {
            case 7:
                System.out.println("Lucky no. 7");
                break;
            default:
                System.out.println("DEFAULT");
        }
    }
}
```

Options

A. char var = '7';

B. char var = 7;

C. Integer var = 7;

D. Character var = '7';

E. Character var = 7;

Answer

A. char var = '7';

[✓ B. char var = 7;]

[✓ C. Integer var = 7;]

D. Character var = '7';

[✓ E. Character var = 7;]

Answer

A. char var = '7';

✓ B. char var = 7;

✓ C. Integer var = 7;

D. Character var = '7';

✓ E. Character var = 7;

switch can accept primitive types: byte, short, int, char; wrapper types: Byte, Short, Integer, Character; String and enums.
In this case, all are valid values but only 3 executes "case 7:". case is comparing integer value 7. NOTE: character seven, '7' is different from integer value seven, 7.
So "char var = '7';" and "Character var = '7';" will print DEFAULT on to the console.

4

What will be the result of compiling and executing Test class?

```
public class Test {
    public static void main(String[] args) {
        double price = 90000;
        String model;
        if(price > 100000) {
            model = "Tesla Model X";
        } else if(price <= 100000) {
            model = "Tesla Model S";
        }
        System.out.println(model);
    }
}
```

Options

A. Tesla Model X
B. Tesla Model
C. null
D. Compilation Error

Answer

A. Tesla Model X

B. Tesla Model

C. null

[✓ D. Compilation Error]

Answer

A. Tesla Model X

B. Tesla Model

C. null

In this case "if - else if" block is used and not "if - else" block.
90000 is assigned to variable 'price' but you can assign parameter value or call some method returning double value, such as:
'double price = currentTemp();'.
In these cases compiler will not know the exact value until runtime, hence Java Compiler is not sure which boolean expression will be evaluated to true and so variable model may not be initialized.

Usage of LOCAL variable, 'model' without initialization gives compilation error.
Hence, System.out.println(model); gives compilation error.

5

What will be the result of compiling and executing Test class?

```
public class Test {
    private static void m(int x) {
        System.out.println("int version");
    }

    private static void m(char x) {
        System.out.println("char version");
    }

    public static void main(String [] args) {
        int i = '5';
        m(i);
        m('5');
    }
}
```

Options

A.	int version int version	**B.**	char version char version
C.	int version char version	**D.**	char version int version
E.	Compilation error		

Answer

A.	int version int version	B.	char version char version
[✓ C.]	int version char version	D.	char version int version
E.	Compilation error		

Answer

A.	int version int version	B.	char version char version
[✓ C.]	int version char version	D.	char version int version
E.	Compilation error		

Method m is overloaded. Which overloaded method to invoke is decided at compile time. m(i) is tagged to m(int) as i is of int type and m('5') is tagged to m(char) as '5' is char literal.

What will be the result of compiling and executing Test class?

```java
package com.udayan.oca;

class Student {
    String name;
    int age;

    void Student() {
        Student("James", 25);
    }

    void Student(String name, int age) {
        this.name = name;
        this.age = age;
    }
}

public class Test {
    public static void main(String[] args) {
        Student s = new Student();
        System.out.println(s.name + ":" + s.age);
    }
}
```

Options

A. Compilation error

B. null:0

C. James:25

D. An exception is thrown at runtime

Answer

A. Compilation error

[✓ B. null:0]

C. James:25

D. An exception is thrown at runtime

6

Answer

A. Compilation error

[✓ B. null:0]

C. James:25

Methods can have same name as the class. Student() and Student(String, int) are methods and not constructors of the class, note the void return type of these methods.
As no constructors are provided in the Student class, java compiler adds default no-arg constructor. That is why the statement Student s = new Student(); doesn't give any compilation error.

Default values are assigned to instance variables, hence null is assigned to name and 0 is assigned to age.

In the output, null:0 is displayed.

7

What will be the result of compiling and executing Test class?

```
//A.java
package com.udayan.oca;

public class A {
    public void print() {
        System.out.println("A");
    }
}


//B.java
package com.udayan.oca;

public class B extends A {
    public void print() {
        System.out.println("B");
    }
}
```

```
//Test.java
package com.udayan.oca.test;

import com.udayan.oca.*;

public class Test {
    public static void main(String[] args) {
        A obj1 = new A();
        B obj2 = (B)obj1;
        obj2.print();
    }
}
```

Options

A. A

B. B

C. Compilation error

D. ClassCastException is thrown at runtime

Answer

A. A

B. B

C. Compilation error

[✓ D. ClassCastException is thrown at runtime]

Answer

A. A

B. B

Class A and B are declared public and inside same package com.udayan.oca.
Method print() of class A has correctly been overridden by B.
print() method is public so no issues in accessing it anywhere.

Let's check the code inside main method.
A obj1 = new A(); => obj1 refers to an instance of class A.
B obj2 = (B)obj1; => obj1 is of type A and it is assigned to obj2 (B type), hence explicit casting is necessary. obj1 refers to an instance of class A, so at runtime obj2 will also refer to an instance of class A. sub type can't refer to an object of super type so at runtime B obj2 = (B)obj1; will throw ClassCastException.

8

What will be the result of compiling and executing Test class?

```
public class Test {
    public static void main(String[] args) {
        try {
            main(args);
        } catch (Exception ex) {
            System.out.println("CATCH-");
        }
        System.out.println("OUT");
    }
}
```

Options

A. CATCH-OUT

B. OUT

C. None of the System.out.println statements are executed

D. Compilation error

Answer

A. CATCH-OUT

B. OUT

[✓ C. None of the System.out.println statements are executed]

D. Compilation error

8

Answer

A. CATCH-OUT

B. OUT

[✓ C. None of the System.out.println statements are]

main(args) method is invoked recursively without specifying any exit condition, so this code ultimately throws java.lang.StackOverflowError. StackOverflowError is a subclass of Error type and not Exception type, hence it is not handled. Stack trace is printed to the console and program ends abruptly.

Java doesn't allow to catch specific checked exceptions if these are not thrown by the statements inside try block.
catch(java.io.FileNotFoundException ex) {} will cause compilation error in this case as main(args); will never throw FileNotFoundException. But Java allows to catch Exception type, hence catch (Exception ex) {} doesn't cause any compilation error.

9

A bank's swift code is generally of 11 characters and used in international money transfers.

An example of swift code: ICICINBBRT4
ICIC: First 4 letters for bank code
IN: Next 2 letters for Country code
BB: Next 2 letters for Location code
RT4: Next 3 letters for Branch code

Which of the following code correctly extracts country code from the swift code referred by String reference variable swiftCode?

Options
A. swiftCode.substring(4, 6);
B. swiftCode.substring(5, 6);
C. swiftCode.substring(5, 7);
D. swiftCode.substring(4, 5);

9

A bank's swift code is generally of 11 characters and used in international money transfers.

An example of swift code: ICICINBBRT4
ICIC: First 4 letters for bank code
IN: Next 2 letters for Country code
BB: Next 2 letters for Location code
RT4: Next 3 letters for Branch code

Which of the following code correctly extracts country code from the swift code referred by String reference variable swiftCode?

Options

✓ A. swiftCode.substring(4, 6);
B. swiftCode.substring(5, 6);
C. swiftCode.substring(5, 7);
D. swiftCode.substring(4, 5);

9

Which of the following code correctly extracts country code from the swift code referred by String reference variable swiftCode?

[✓ A.] swiftCode.substring(4, 6);
B. swiftCode.substring(5, 6);
C. swiftCode.substring(5, 7);
D. swiftCode.substring(4, 5);

substring(int beginIndex, int endIndex) is used to extract the substring. The substring begins at "beginIndex" and extends till "endIndex - 1".
Country code information is stored at index 4 and 5, so the correct substring method to extract country code is:
swiftCode.substring(4, 6);

What will be the result of compiling and executing Test class?

```
//Test.java
package com.udayan.oca;

class SpecialString {
    String str;
    SpecialString(String str) {
        this.str = str;
    }
}

public class Test {
    public static void main(String[] args) {
        System.out.println(new String("Java"));
        System.out.println(new StringBuilder("Java"));
        System.out.println(new SpecialString("Java"));
    }
}
```

Options

A.	Java Java Java	**B.**	Java Java <Some text containing @ symbol>
C.	Java <Some text containing @ symbol> <Some text containing @ symbol>	**D.**	Compilation error

Answer

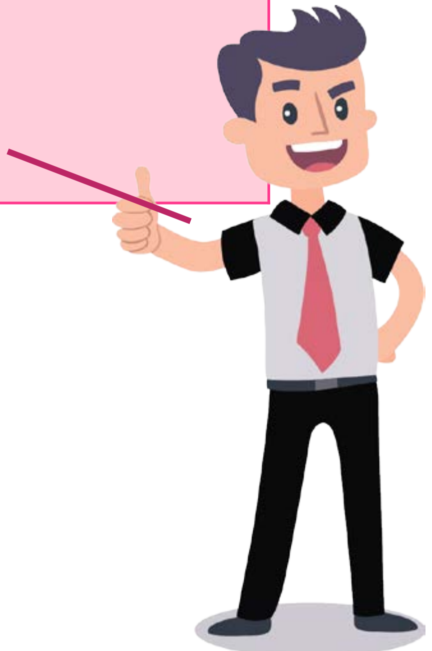
A.	Java Java Java	[✓ B.]	Java Java <Some text containing @ symbol>
C.	Java <Some text containing @ symbol> <Some text containing @ symbol>	D.	Compilation error

10

Answer

A.	Java Java Java	[✓ B.]	Java Java <Some text containing @ symbol>
C.	Java <Some text containing @ symbol> <Some text containing @ symbol>	D.	Compilation error

String and StringBuilder classes override toString() method, which prints the text stored in these classes. SpecialString class doesn't override toString() method and hence when instance of SpecialString is printed on to the console, you get: <fully qualified name of SpecialString class>@<hexadecimal representation of hashcode>.



What will be the result of compiling and executing Test class?

```
//Test.java
import java.time.LocalDate;

public class Test {
    public static void main(String [] args) {
        LocalDate newYear = LocalDate.of(2018, 1, 1);
        LocalDate christmas = LocalDate.of(2018, 12, 25);
        boolean flag1 = newYear.isAfter(christmas);
        boolean flag2 = newYear.isBefore(christmas);
        System.out.println(flag1 + ":" + flag2);
    }
}
```

Options

A. false:true
B. true:false
C. An exception is thrown at runtime
D. Compilation error

Answer

[✓ A. false:true]

B. true:false

C. An exception is thrown at runtime

D. Compilation error

Answer

[✓ A. false:true]

B. true:false

C. An exception is thrown at runtime

D. Compilation error

isAfter and isBefore method can be interpreted as:
Does 1st Jan 2018 come after 25th Dec 2018? No, false. Does 1st Jan 2018 come before 25th Dec 2018? Yes, true.

12

What will be the result of compiling and executing Test class?

```
//Test.java
import java.time.LocalDate;
import java.time.Period;
import java.time.format.DateTimeFormatter;

public class Test {
    public static void main(String [] args) {
        LocalDate date = LocalDate.of(2012, 1, 11);
        Period period = Period.ofMonths(2);
        DateTimeFormatter formatter =
                DateTimeFormatter.ofPattern("MM-dd-yy");
        System.out.print(formatter.format(date.minus(period)));
    }
}
```

Options

A. 11-11-12

B. 01-11-11

C. 01-11-12

D. 11-11-11

E. Runtime exception

Answer

A. 11-11-12

B. 01-11-11

C. 01-11-12

[✓ D. 11-11-11]

E. Runtime exception

Answer

A. 11-11-12

B. 01-11-11

C. 01-11-12

[✓ D. 11-11-11]

E. Runtime exception

date --> {2012-01-11}, period --> {P2M}, date.minus(period) --> {2011-11-11} [subtract 2 months period from {2012-01-11}, month is changed to 11 and year is changed to 2011].

formatter -> {MM-dd-yy}, when date {2011-11-11} is formatter in this format 11-11-11 is printed on to the console.

What will be the result of compiling and executing Test class?

```
import java.util.function.Predicate;

public class Test {
    public static void main(String[] args) {
        String [] arr = {"A", "ab", "bab", "Aa", "bb",
                            "baba", "aba", "Abab"};

        Predicate<String> p = s ->
                s.toUpperCase().substring(0,1).equals("A");

        processStringArray(arr, p);
    }

    private static void processStringArray(String [] arr, Predicate<String> predicate) {
        for(String str : arr) {
            if(predicate.test(str)) {
                System.out.println(str);
            }
        }
    }
}
```

Options

A.	Compilation error	**B.**	Runtime exception
C.	A Aa Abab	**D.**	ab aba
E.	A ab Aa aba Abab		

Answer

A.	Compilation error	**B.**	Runtime exception
C.	A Aa Abab	**D.**	ab aba
✓ **E.**	A ab Aa aba Abab		

Answer

A.	Compilation error	B.	Runtime exception
C.	A Aa	D.	ab aba

Let us suppose test string is "aba".
Lambda expression
s.toUpperCase().substring(0,1).equals("A"); means:
"aba".toUpperCase().substring(0,1).equals("A"); =>
"ABA".substring(0,1).equals("A"); => "A".equals("A"); => true.

This lambda expression returns true for any string starting with a (in lower or upper case). Based on the lambda expression, 5 array elements passes the Predicate's test and are printed on to the console.

14

What will be the result of compiling and executing Test class?

```
public class Test {
    public static void main(String[] args) {
        Boolean b1 = new Boolean("tRuE");
        Boolean b2 = new Boolean("fAlSe");
        Boolean b3 = new Boolean("abc");
        Boolean b4 = null;
        System.out.println(b1 + ":" + b2 + ":" + b3 + ":" + b4);
    }
}
```

Options

A. false:false:false:null
B. true:false:false:null
C. false:false:true:null
D. Compilation error

Answer

A. false:false:false:null

[✓ B. true:false:false:null]

C. false:false:true:null

D. Compilation error

Answer

A. false:false:false:null

[✓ B. true:false:false:null]

C. false:false:true:null

D. Compilation error

Boolean class code uses equalsIgnoreCase method to validate the passed String, so if passed String is "true" ('t', 'r', 'u' and 'e' can be in any case), then boolean value stored in Boolean object is true otherwise false.

b1 stores true, b2 stores false, b3 stores false and as b4 is of reference type, hence it can store null as well.
Output is: true:false:false:null

15

What will be the result of compiling and executing Test class?

```
public class Test {

    private static void add(double d1, double d2) {
        System.out.println("double version: " + (d1 + d2));
    }

    private static void add(Double d1, Double d2) {
        System.out.println("Double version: " + (d1 + d2));
    }

    public static void main(String[] args) {
        add(10.0, null);
    }

}
```

Options

A. Compilation error
B. double version: 10.0
C. Double version: 10.0
D. An exception is thrown at runtime

Answer

A. Compilation error

B. double version: 10.0

C. Double version: 10.0

[✓ D. An exception is thrown at runtime]

Answer

A. Compilation error

B. double version: 10.0

C. Double version: 10.0

[✓ D.] An exception is thrown at runtime

add(10.0, null); => Compiler can't convert null to double primitive type, so 2nd argument is tagged to Double reference type. So to match the method call, 10.0 is converted to Double object by auto-boxing and add(10.0, null); is tagged to add(Double, Double); method.

But at the time of execution, d2 is null so System.out.println("Double version: " + (d1 + d2)); throws NullPointerException.

What will be the result of compiling and executing Test class?

```
public class Test {
    public static void main(String[] args) {
        int score = 60;
        switch (score) {
            default:
                System.out.println("Not a valid score");
            case score < 70:
                System.out.println("Failed");
                break;
            case score >= 70:
                System.out.println("Passed");
                break;
        }
    }
}
```

Options

A.	Compilation error	**B.**	Failed
C.	Not a valid score Failed	**D.**	Passed

Answer

✓ **A.**	Compilation error	**B.**	Failed
C.	Not a valid score Failed	**D.**	Passed

Answer

✓ A.	Compilation error	B.	Failed

case values must evaluate to the same type / compatible type as the switch expression can use.
switch expression can accept following:
char or Character,
byte or Byte,
short or Short,
int or Integer,
An enum only from Java 6,
A String expression only from Java 7.

In this case, switch expression [switch (score)] is of int type.
But case expressions, score < 70 and score >= 70 are of boolean type and hence compilation error.

What will be the result of compiling and executing Test class?

```
public class Test {
    public static void main(String [] args) {
        int a = 2;
        boolean res = false;
        res = a++ == 2 || --a == 2 && --a == 2;
        System.out.println(a);
    }
}
```

Options

A. 2

B. 3

C. 1

D. Compilation error

17

Answer

A. 2

[✓ B. 3]

C. 1

D. Compilation error

Answer

A. 2

[✓ B. 3]

a++ == 2 || --a == 2 && --a == 2; [Given expression].
(a++) == 2 || --a == 2 && --a == 2; [Postfix has got higher precedence than other operators].
(a++) == 2 || (--a) == 2 && (--a) == 2; [After postfix, precedence is given to prefix].
((a++) == 2) || ((--a) == 2) && ((--a) == 2); [== has higher precedence over && and ||].
((a++) == 2) || (((--a) == 2) && ((--a) == 2)); [&& has higher precedence over ||].
Let's start solving it:
((a++) == 2) || (((--a) == 2) && ((--a) == 2)); [a=2, res=false].
(2 == 2) || (((--a) == 2) && ((--a) == 2)); [a=3, res=false].
true || (((--a) == 2) && ((--a) == 2)); [a=3, res=false]. || is a short-circuit operator, hence no need to evaluate expression on the right.
res is true and a is 3.

18

What will be the result of compiling and executing Test class?

```
class Vehicle {
    public int getRegistrationNumber() {
        return 1;
    }
}

class Car {
    public int getRegistrationNumber() {
        return 2;
    }
}

public class Test {
    public static void main(String[] args) {
        Vehicle obj = new Car();

System.out.println(obj.getRegistrationNumber());
    }
}
```

Options

A. 1
B. 2
C. An exception is thrown at runtime
D. Compilation error

Answer

A. 1

B. 2

C. An exception is thrown at runtime

[✓ D. Compilation error]

Answer

A. 1

B. 2

C. An exception is thrown at runtime

[✓ D. Compilation error]

class Car doesn't extend from Vehicle class, this means Vehicle is not super type of Car.
Hence, Vehicle obj = new Car(); gives compilation error.

19

What will be the result of compiling and executing Test class?

```
public class Test {
    private static void m1() throws Exception {
        throw new Exception();
    }

    public static void main(String[] args) {
        try {
            m1();
        } finally {
            System.out.println("A");
        }
    }
}
```

Options

A. A is printed to the console and program ends normally.

B. A is printed to the console, stack trace is printed and then program ends normally.

C. A is printed to the console, stack trace is printed and then program ends abruptly.

D. Compilation error.

Answer

A. A is printed to the console and program ends normally.

B. A is printed to the console, stack trace is printed and then program ends normally.

C. A is printed to the console, stack trace is printed and then program ends abruptly.

[✓ D. Compilation error.]

Answer

A. A is printed to the console and program ends normally.

B. A is printed to the console, stack trace is printed and then program ends normally.

C. A is printed to the console, stack trace is printed and then program ends abruptly.

Method m1() throws Exception (checked) and it declares to throw it, so no issues with method m1().
But main() method neither provides catch handler nor throws clause and hence main method gives Compilation error.
Handle or Declare rule should be followed for checked exception if you are not re-throwing it.

20

Consider below interface declaration:

```
public interface I1 {
    void m1() throws java.io.IOException;
}
```

Which of the following incorrectly implements interface I1?

A.	`public class C1 implements I1 {`
	`    public void m1() {}`
	`}`

B.	`public class C2 implements I1 {`
	`    public void m1() throws java.io.FileNotFoundException{}`
	`}`

C.	`public class C3 implements I1 {`
	`    public void m1() throws java.io.IOException{}`
	`}`

D.	`public class C4 implements I1 {`
	`    public void m1() throws Exception{}`
	`}`

20

Consider below interface declaration:

```
public interface I1 {
    void m1() throws java.io.IOException;
}
```

Which of the following incorrectly implements interface I1?

A.	`public class C1 implements I1 {`
	`    public void m1() {}`
	`}`

B.	`public class C2 implements I1 {`
	`    public void m1() throws java.io.FileNotFoundException{}`
	`}`

C.	`public class C3 implements I1 {`
	`    public void m1() throws java.io.IOException{}`
	`}`

✓ D.	`public class C4 implements I1 {`
	`    public void m1() throws Exception{}`
	`}`

Answer

A.

```
public class C1 implements I1 {
    public void m1() {}
}
```

oundException{}

ion{}

NOTE: Question is asking for "incorrect" implementation and not "correct" implementation.

According to overriding rules, if super class / interface method declares to throw a checked exception, then overriding method of sub class / implementer class has following options:
1. May not declare to throw any checked exception,
2. May declare to throw the same checked exception thrown by super class / interface method,
3. May declare to throw the sub class of the exception thrown by super class / interface method,
4. Cannot declare to throw the super class of the exception thrown by super class / interface method

21

What will be the result of compiling and executing Test class?

```
import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("X");
        list.add("Y");
        list.add("X");
        list.add("Y");
        list.add("Z");
        list.remove(new String("Y"));
        System.out.println(list);
    }
}
```

Options

A. [X, X, Y, Z]
B. [X, X, Z]
C. [X, Z]
D. [X, Y, Z]
E. Compilation error
F. Exception is thrown at runtime

Answer

- [✓ A.] [X, X, Y, Z]
- B. [X, X, Z]
- C. [X, Z]
- D. [X, Y, Z]
- E. Compilation error
- F. Exception is thrown at runtime

Answer

✓ A. [X, X, Y, Z]

B. [X, X, Z]

C. [X, Z]

D. [X, Y, Z]

After all the add statements are executed, list contains: [X, Y, X, Y, Z].
list.remove(new String("Y")); removes the first occurrence of "Y" from the list, which means the 2nd element of the list.
After removal list contains: [X, X, Y, Z].

NOTE: String class and all the wrapper classes override equals(Object) method, hence at the time of removal when another instance is passed [new String("Y")], there is no issue in removing the matching item.

22

What will be the result of compiling and executing Test class?

```
//Test.java
import java.time.Period;

public class Test {
    public static void main(String [] args) {
        Period period = Period.
                of(2, 1, 0).ofYears(10).ofMonths(5).ofDays(2);
        System.out.println(period);
    }
}
```

Options

A. P12Y6M2D

B. P2Y1M0D

C. P2Y1M

D. P2D

Answer

A. P12Y6M2D

B. P2Y1M0D

C. P2Y1M

[✓ D. P2D]

Answer

A. P12Y6M2D

B. P2Y1M0D

C. P2Y1M

[✓ D. P2D]

of and ofXXX methods are static methods and not instance methods.
Period.of(2, 1, 0) => returns an instance of Period type. static methods can be invoked using class_name or using reference variable.
In this case ofYears(10) is invoked on period instance but compiler uses Period's instance to resolve the type, which is period. A new Period instance {P10Y} is created, after that another Period instance {P5M} is created and finally Period instance {P2D} is created. This instance is assigned to period reference variable and hence P2D is printed on to the console.

What will be the result of compiling and executing Test class?

```
import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String [] args) {
        List<Integer> list = new ArrayList<Integer>();

        list.add(27);
        list.add(27);

        list.add(new Integer(27));
        list.add(new Integer(27));

        System.out.println(list.get(0) == list.get(1));
        System.out.println(list.get(2) == list.get(3));
    }
}
```

Options

A.	false false	**B.**	false true
C.	true true	**D.**	true false

23

Answer

A.	false false	**B.**	false true
C.	[✓ true true]	**D.**	true false

Answer

This is bit tricky. Just remember this:
Two instances of following wrapper objects, created through auto-boxing will always be same, if their primitive values are same:
Boolean,
Byte,
Character from \u0000 to \u007f (7f equals to 127),
Short and Integer from -128 to 127.

For 1st statement, list.add(27); => Auto-boxing creates an integer object for 27.
For 2nd statement, list.add(27); => Java compiler finds that there is already an Integer object in the memory with value 27, so it uses the same object.

That is why System.out.println(list.get(0) == list.get(1)); returns true.

new Integer(27) creates a new Object in the memory, so
System.out.println(list.get(2) == list.get(3)); returns false.

What will be the result of compiling and executing Test class?

```
import java.util.function.Predicate;

public class Test {
    public static void main(String[] args) {
        String [] arr = {"*", "**", "***", "****", "*****"};
        Predicate pr1 = s -> s.length() < 4;
        print(arr, pr1);
    }

    private static void print(String [] arr,
Predicate<String> predicate) {
        for(String str : arr) {
            if(predicate.test(str)) {
                System.out.println(str);
            }
        }
    }
}
```

24

Options

A.	Compilation error	**B.**	* ** ***
C.	* ** *** ****	**D.**	* ** *** **** *****

Answer

✓ A.	Compilation error	B.	* ** ***
C.	* ** *** ****	D.	* ** *** **** *****

Answer

[✓ A.]	Compilation error	B.	* ** ***
C.	* **	D.	* **

Though Predicate is a generic interface but raw type is also allowed. Type of the variable in lambda expression is inferred by the generic type of Predicate<T> interface.

In this case, Predicate pr1 = s -> s.length() < 4; Predicate is considered of Object type so variable "s" is of Object type and Object class doesn't have length() method. So, s.length() causes compilation error.

25

What will be the result of compiling and executing Test class?

```
import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<>();
        list.add(110);
        list.add(new Integer(110));
        list.add(110);

        list.removeIf(i -> i == 110);
        System.out.println(list);
    }
}
```

Options

A. [110, 110, 110]

B. [110, 110]

C. [110]

D. []

Answer

A. [110, 110, 110]

B. [110, 110]

C. [110]

[✓ D. []]

Answer

As list can store only wrapper objects and not primitives, hence for list.add(110); auto-boxing creates an Integer object {110}.

For list.add(new Integer(110)); as new keyword is used so another Integer object {110} is created.

For 3rd add method call, list.add(110); auto-boxing kicks in and as 110 is between -128 to 127, hence Integer object created at 1st statement is referred.

removeIf(Predicate) method was added as a default method in Collection interface in JDK 8 and it removes all the elements of this collection that satisfy the given predicate.

Boolean expression is : i == 110; in this expression i is wrapper object and 110 is int literal so java extracts int value of wrapper object, i and then equates. As all the 3 objects store 110, hence true is returned. All integer objects are removed from the list.

If list.removeIf(i -> i == new Integer(110)); was used, then all three list elements would return false as object references are equated and not contents.

OCA — 1Z0-808
Exam Practice Test