OCP — 1Z0-809

Oracle Certified Associate, Java SE 8 Programmer II

Are You ready for OCP Exam?

Do you want to check your preparation?

Then you have come to the right place

OCP — 1Z0-809

Oracle Certified Associate, Java SE 8 Programmer II

Never Seen before Questions

25 Practice Test Questions with Explanation

Extracted from Highest Rated Course on Udemy

```
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: " +
}
```

OCP — 1Z0-809
Exam Practice Test

Hello Everyone
My Name is Udayan Khattry
This slideshow is an excerpt from my popular course OCA-1Z0-809 Practice Test on Udemy which has helped many 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..

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

1

```
package com.udayan.ocp;

enum Flags {
    TRUE, FALSE;

    Flags() {
System.out.println("HELLO");
    }
}

public class Test {
    public static void
main(String[] args) {
        Flags flags = new
Flags();
    }
}
```

1

Options

A. HELLO is printed twice.

B. HELLO is printed once.

C. Exception is thrown at runtime.

D. None of the other options.

1

Answer

A. HELLO is printed twice.

B. HELLO is printed once.

C. Exception is thrown at runtime.

[✓ D. None of the above options.]

1

Answer

A. HELLO is printed twice.

B. HELLO is printed once.

C. Exception is thrown at runtime.

[✓ D. None of the above options.]

new Flags() tries to invoke enum constructor but Enum constructor cannot be invoked.

2

Consider given code

```
package com.udayan.ocp;

class Outer {
    private String name = "James Gosling";
    //Insert inner class definition here
}

public class Test {
    public static void main(String [] args) {
        new Outer().new Inner().printName();
    }
}
```

Which of the following Inner class definitions, if inserted in the Outer class, will print 'James Gosling' in the output on executing Test class?

2

Options

A.

```
class Inner {
    public void printName() {
        System.out.println(this.name);
    }
}
```

B.

```
class Inner {
    public void printName() {
        System.out.println(name);
    }
}
```

C.

```
inner class Inner {
    public void printName() {
        System.out.println(name);
    }
}
```

D.

```
abstract class Inner {
    public void printName() {
        System.out.println(name);
    }
}
```

2

Answer

[✓ A.]	class Inner {
	    public void printName() {
	        System.out.println(this.name);
	    }
	}
		B.	class Inner {
			    public void printName() {
			        System.out.println(name);
			    }
			}

C.	inner class Inner {
	    public void printName() {
	        System.out.println(name);
	    }
	}
		D.	abstract class Inner {
			    public void printName() {
			        System.out.println(name);
			    }
			}

2

Answer

A.	class Inner {
	    public void printName() {
	        System.out.println(this.name);
		✓ B.	class Inner {
			    public void printName() {
			        System.out.println(name);

name can be referred either by name or Outer.this.name. There is no keyword with the name 'inner' in java.

As new Inner() is used in main method, hence cannot declare class Inner as abstract in this case. But note abstract or final can be used with regular inner classes.

Keyword 'this' inside Inner class refers to currently executing instance of Inner class and not the Outer class.

To access Outer class variable from within inner class you can use these 2 statements: System.out.println(name); OR

System.out.println(Outer.this.name);

3

What is the purpose of below lambda expression?

```
(x, y) -> x + y;
```

Options

A. It accepts two int arguments, adds them and returns the int value
B. It accepts two String arguments, concatenates them and returns the String instance
C. It accepts a String and an int arguments, concatenates them and returns the String instance
D. Not possible to define the purpose

3

Answer

A. It accepts two int arguments, adds them and returns the int value

B. It accepts two String arguments, concatenates them and returns the String instance

C. It accepts a String and an int arguments, concatenates them and returns the String instance

[✓ D. Not possible to define the purpose]

3

Answer

A. It accepts two int arguments, adds them and returns the int value

B. It accepts two String arguments, concatenates them and returns the String instance

C. It accepts a String and an int arguments, concatenates them and returns the String instance

Lambda expression doesn't work without target type and target type must be a functional interface.

In this case as the given lambda expression is not assigned to any target type, hence its purpose is not clear. In fact, given lambda expression gives compilation error without its target type.

4

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

```
package com.udayan.ocp;

class Printer<String> {
    private String t;
    Printer(String t){
        this.t = t;
    }
}
public class Test {
    public static void main(String[] args) {
        Printer<Integer> obj = new Printer<>(100);
        System.out.println(obj);
    }
}
```

4

Options

A. 100

B. Some text containing @ symbol

C. Compilation error in Printer class

D. Compilation error in Test class

4

Answer

A. 100

[✓ B. Some text containing @ symbol]

C. Compilation error in Printer class

D. Compilation error in Test class

4

Answer

A. 100

[✓ B. Some text containing @ symbol]

C. Compilation error in Printer class

Type parameter should not be a Java keyword & a valid Java identifier. Naming convention for Type parameter is to use uppercase single character.
In class Printer<String>, 'String' is a valid Java identifier and hence a valid type parameter even though it doesn't follow the naming convention of uppercase single character.
Printer<Integer> obj = new Printer<>(100); => Type argument is Integer and it correctly creates an instance of Printer class passing Integer object 100.
Printer class doesn't override toString() method and hence 'System.out.println(obj);' prints some text containing @ symbol.

5

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

```
package com.udayan.ocp;

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

public class Test {
    public static void main(String[] args) {
        List<? super String> list = new ArrayList<>();
        list.add("A");
        list.add("B");
        for(String str : list) {
            System.out.print(str);
        }
    }
}
```

5

Options

A. AB

B. Compilation error

C. Runtime exception

5

Answer

A. AB

[✓ B. Compilation error]

C. Runtime exception

Answer

A. AB

[✓ B. Compilation error]

C. Runtime exception

For 'List<? super String>' type of read objects is 'Object' and type of write objects is 'String' and its subclasses (no subclass of String as String is final).

'for(String str : list)' causes compilation failure. Correct syntax should be: 'for(Object str : list)'

6

Consider given code

```
package com.udayan.ocp;

public class Test {
    public static void main(String[] args) {
        Operation o1 = (x, y) -> x + y;
        System.out.println(o1.operate(5, 10));
    }
}
```

Which of the following functional interface definitions can be used here, so that the output of above code is: 15? Select ALL that apply.

Options

A.	interface Operation {
	    int operate(int x, int y);
	}
		B.	interface Operation {
			    long operate(long x, long y);
			}

C.	interface Operation<T> {
	    T operate(T x, T y);
	}
		D.	interface Operation<T extends
			Integer> {
			    T operate(T x, T y);
			}

Answer

[✓ A.]	interface Operation {
	int operate(int x, int y);
	}
		[✓ B.]	interface Operation {
			long operate(long x, long y);
			}

C.	interface Operation<T> {
	T operate(T x, T y);
	}
		[✓ D.]	interface Operation<T extends
			Integer> {
			T operate(T x, T y);
			}

6

Answer

A. ✓

```
interface Operation {
   int operate(int x, int y);
```

B. ✓

```
interface Operation {
   long operate(long x, long y);
```

From the given syntax inside main method, it is clear that interface name is Operation and it has an abstract method operate which accepts 2 parameters of numeric type and returns the numeric result (as result of adding 5 and 10 is 15). So, int and long versions can be easily applied here.

Operation<T> will not work here as inside main method, raw type is used, which means x and y will be of Object type and x + y will give compilation error as + operator is not defined when both the operands are of Object type.

For Operation<T extends Integer>, even though main method uses raw type, but x and y will be of Integer type and hence x + y will not give any compilation error.

6

Answer

A. Compilation error

[✓ B. null:0]

C. James:25

From the given syntax inside main method, it is clear that interface name is Operation and it has an abstract method operate which accepts 2 parameters of numeric type and returns the numeric result (as result of adding 5 and 10 is 15). So, int and long versions can be easily applied here.

Operation<T> will not work here as inside main method, raw type is used, which means x and y will be of Object type and x + y will give compilation error as + operator is not defined when both the operands are of Object type.

For Operation<T extends Integer>, even though main method uses raw type, but x and y will be of Integer type and hence x + y will not give any compilation error.

7

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

```
package com.udayan.ocp;
import java.util.ArrayDeque;
import java.util.Deque;

public class Test {
    public static void main(String[] args) {
        Deque<Boolean> deque = new ArrayDeque<>();
        deque.push(new Boolean("abc"));
        deque.push(new Boolean("tRuE"));
        deque.push(new Boolean("FALSE"));
        deque.push(true);
        System.out.println(deque.pop() + ":"                +
deque.peek() + ":" + deque.size());
    }
}
```

7

Options

A. true:true:3

B. false:false:3

C. false:true:3

D. true:false:3

7

Answer

A. true:true:3

B. false:false:3

C. false:true:3

[✓ D. true:false:3]

7

Answer

push, pop and peek are Stack's terminology.
push(E) calls addFirst(E), pop() calls removeFirst() and peek() invokes peekFirst(), it just retrieves the first element (HEAD) but doesn't remove it.

deque.push(new Boolean("abc")); => [*false]. * represents HEAD of the deque.
deque.push(new Boolean("tRuE")); => [*true, false].
deque.push(new Boolean("FALSE")); => [*false, true, false].
deque.push(true); => [*true, false, true, false].

deque.pop() => removes and returns the HEAD element, true in this case.
deque => [*false, true, false].

deque.peek() => retrieves but doesn't remove the HEAD element, false in this case. deque => [*false, true, false].

deque.size() => 3. Hence output is 'true:false:3'.

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

```
package com.udayan.ocp;
import java.util.function.Consumer;

public class Test {
    public static void main(String[] args) {
        Consumer<Integer> consumer = System.out::print;
        Integer i = 5;
        consumer.andThen(consumer).accept(i++); //Line 7
    }
}
```

8

Options

A. 55
B. 56
C. 66
D. Compilation error

8

Answer

[✓ A. 55]

B. 56

C. 66

D. Compilation error

8

Answer

[✓ A. 55]

B. 56

C. 66

andThen is the default method defined in Consumer interface, so it is invoked on consumer reference variable.

Value passed in the argument of accept method is passed to both the consumer objects. So, for understanding purpose Line 7 can be split into: consumer.accept(5); consumer.accept(5); So it prints '55' on to the console.

Check the code of andThen method defined in Consumer interface to understand it better.

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

```
package com.udayan.ocp;
import java.util.NavigableMap;
import java.util.TreeMap;
import java.util.function.BiConsumer;

public class Test {
    public static void main(String[] args) {
        NavigableMap<Integer, String> map = new TreeMap<>();
        BiConsumer<Integer, String> consumer = map::putIfAbsent;
        consumer.accept(1, null);
        consumer.accept(2, "two");
        consumer.accept(1, "ONE");
        consumer.accept(2, "TWO");

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

9

Options

A. {1=null, 2=two}
B. {1=ONE, 2=TWO}
C. {1=ONE, 2=two}
D. {1=null, 2=two}
E. {1=null, 2=TWO}

9

Answer

A. {1=null, 2=two}

B. {1=ONE, 2=TWO}

[✓ C. {1=ONE, 2=two}]

D. {1=null, 2=two}

E. {1=null, 2=TWO}

9

Answer

A. {1=null, 2=two}

Though reference variable of NavigableMap is used but putIfAbsent method is from Map interface. It is a default method added in JDK 8.0.
BiConsumer<T, U> : void accept(T t, U u);
Lambda expression corresponding to 'map::putIfAbsent;' is '(i, s) -> map.putIfAbsent(i, s)'
This is the case of "Reference to an Instance Method of a Particular Object".
TreeMap sorts the data on the basis of natural order of keys.

consumer.accept(1, null); => {1=null}.
consumer.accept(2, "two"); => {1=null, 2=two}.
consumer.accept(1, "ONE"); => {1=ONE, 2=two}. putIfAbsent method replaces null value with the new value.
consumer.accept(2, "TWO"); => {1=ONE, 2=two}. As value is available against '2', hence value is not replaced.

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

```
package com.udayan.ocp;

import java.util.Arrays;
import java.util.List;

public class Test {
    public static void main(String[] args) {
        List<String> codes = Arrays.asList("1st",
                                "2nd", "3rd", "4th");
        System.out.println(codes.stream().filter(s ->
            s.endsWith("d")).reduce((s1, s2) -> s1 + s2));
    }
}
```

10

Options

A. 1st4th

B. 2nd3rd

C. Optional[1st4th]

D. Optional[2nd3rd]

10

Answer

A. 1st4th

B. 2nd3rd

C. Optional[1st4th]

[✓ D. Optional[2nd3rd]]

10

Answer

A. 1st4th

B. 2nd3rd

C. Optional[1st4th]

[✓ D. Optional[2nd3rd]]

filter method filters all the strings ending with "d".

'stream.reduce((s1, s2) -> s1 + s2)' returns 'Optional<String>' type whereas 'stream.reduce("", (s1, s2) -> s1 + s2)' returns 'String'.

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

```
package com.udayan.ocp;

import java.util.stream.IntStream;

public class Test {
    public static void main(String[] args) {
        int sum = IntStream.rangeClosed(1,3).map(i -> i * i)
                .map(i -> i * i).sum();
        System.out.println(sum);
    }
}
```

11

Options

A. 6

B. 14

C. 98

D. None of the above options

11

Answer

A. 6

B. 14

[✓ C. 98]

D. None of the above options

11

Answer

A. 6

IntStream.rangeClosed(int start, int end) => Returns a sequential stream from start to end, both inclusive and with a step of 1.

IntStream.map(IntUnaryOperator) => Returns a stream consisting of the results of applying the given function to the elements of this stream.

IntStream.rangeClosed(1,3) => [1,2,3].
map(i -> i * i) => [1,4,9].
map(i -> i * i) => [1,16,81].
sum() => 1+16+81 = 98.

12 What will be the result of compiling and executing Test class?
User input is: HELLO

```
package com.udayan.ocp;

import java.util.Scanner;

public class Test {
    public static void main(String[] args) {
        System.out.print("Enter some text: ");
        try(Scanner scan = new Scanner(System.in)) {
            String s = scan.nextLine();
            System.out.println(s);
            scan.close();
            scan.close();
        }
    }
}
```

12

Options

A. Compilation error

B. Runtime Exception

C. On execution program terminates successfully after printing 'HELLO' on to the console

12

Answer

A. Compilation error

B. Runtime Exception

[✓ C. On execution program terminates successfully after printing 'HELLO' on to the console]

12

Answer

A. Compilation error

B. Runtime Exception

C. On execution program terminates successfully

Even though Scanner is created in try-with-resources block, calling close() method explicitly doesn't cause any problem. Scanner class allows to invoke close() method multiple times. In this case, it will be called 3 times: twice because of scan.close() and once because of try-with-resources statement.

'HELLO' is printed on to the console and program terminates successfully.

12

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.

13 Daylight saving time 2018 in United States (US) ends at 4-Nov-2018 2:00 AM. What will be the result of compiling and executing Test class?

```
package com.udayan.ocp;

import java.time.*;

public class Test {
    public static void main(String [] args) {
        LocalDate date = LocalDate.of(2018, 11, 4);
        LocalTime time = LocalTime.of(13, 59, 59);
        ZonedDateTime dt = ZonedDateTime.of(date,
                time, ZoneId.of("America/New_York"));
        dt = dt.plusSeconds(1);
        System.out.println(dt.getHour() + ":" +
                dt.getMinute() + ":" + dt.getSecond());
    }
}
```

13

Options

A. 12:0:0

B. 13:0:0

C. 14:0:0

13

Answer

A. 12:0:0

B. 13:0:0

[✓ C. 14:0:0]

13

Answer

You should be aware of Day light saving mechanism to answer this question.
Suppose daylight time starts at 2 AM on particular date.
Current time: 1:59:59 [Normal time].
Next second: 3:00:00 [Time is not 2:00:00 rather it is 3:00:00. It is Daylight saving time].
Clock just jumped from 1:59:59 to 3:00:00.

Now Suppose daylight time ends at 2 AM on particular date.
Current time: 1:59:59 [Daylight saving time].
Next second: 1:00:00 [Time is not 2:00:00 rather it is 1:00:00. Clock switched back to normal time].
Clock just went back from 1:59:59 to 1:00:00.

Now let's solve given code:
dt --> {2018-11-04T13:59:59}. Daylight saving time has already ended as it is PM and not AM. This is acturally a normal time now.

dt.plusSeconds(1) => creates a new ZonedDateTime object {2018-11-04T14:00:00} and dt refers to it.

dt.getHour() = 14, dt.getMinute() = 0 and dt.getSecond() = 0.

14 Will below code display time part on to the console?

```
package com.udayan.ocp;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.time.format.FormatStyle;

public class Test {
    public static void main(String [] args) {
        LocalDateTime date = LocalDateTime.of(
                                    2019, 1, 1, 10, 10);
        DateTimeFormatter formatter = DateTimeFormatter.
                    ofLocalizedDate(FormatStyle.FULL);
        System.out.println(formatter.format(date));
    }
}
```

14

Options

A. Yes

B. No

14

Answer

A. Yes

[✓ B. No]

14

Answer

A. Yes

[✓ B. No]

DateTimeFormatter.ofLocalizedDate(FormatStyle.FULL); statement returns formatter to format date part.

date is of LocalDateTime type hence it has both date part and time part.

'formatter.format(date)' simply formats the date part and ignores time part.

NOTE: You should aware of other formatter related methods for the OCP exam, such as: 'ofLocalizedTime' and 'ofLocalizedDateTime'

15

Given code of Test.java file:

```
package com.udayan.ocp;

import java.io.*;
import java.nio.file.Files;
import java.nio.file.Paths;

public class Test {
    public static void main(String[] args)   throws IOException {
      /*INSERT*/
    }
}
```

F: is accessible for reading and contains 'Book.java' file. Which of the following statements, if used to replace /*INSERT*/, will successfully print contents of 'Book.java' on to the console? Select 3 options.

15

Options

A.	Files.*lines*(Paths.*get*("F:\\Book.java")).forEach(System.*out*::println);	**B.**	Files.lines(Paths.get("F:\\Book.java")).stream().forEach(System.out::println);
C.	Files.*readAllLines*(Paths.*get*("F:\\Book.java")).forEach(System.*out*::println);	**D.**	Files.readAllLines(Paths.get("F:\\Book.java")).stream().forEach(System.out::println);

15

Answer

✓ **A.**	Files.*lines*(Paths.*get*("F:\\Book.java")).forEach(System.*out*::println);	**B.**	Files.lines(Paths.get("F:\\Book.java")).stream().forEach(System.out::println);
✓ **C.**	Files.*readAllLines*(Paths.*get*("F:\\Book.java")).forEach(System.*out*::println);	✓ **D.**	Files.readAllLines(Paths.get("F:\\Book.java")).stream().forEach(System.out::println);

15

Answer

A.	Files.*lines*(Paths.*get*("F:\\Book.	B.	Files.lines(Paths.get("F:\\Book.java"
			:prin
			\\Bo
			em.o

Below are the declarations of lines and readAllLines methods from Files class:

public static Stream<String> lines(Path path) throws IOException {...}
public static List<String> readAllLines(Path path) throws IOException {...}

'Files.lines(Paths.get("F:\\Book.java"))' returns Stream<String> object. Hence forEach() can be invoked but stream() can't be invoked.

'Files.readAllLines(Paths.get("F:\\Book.java"))' returns List<String> object. Hence both forEach() and stream() methods can be invoked. List has both the methods. But converting list to stream() and then invoking forEach() method is not required but it is a legal syntax and prints the file contents.

16

C:\ is accessible for reading/writing and below is the content of 'C:\TEMP' folder:

```
C:\TEMP
|   msg
└───Parent
    └───Child
            Message.txt
```

'msg' is a symbolic link file for 'C:\TEMP\Parent\Child\Message.txt'.

Message.txt contains following text:
Welcome!

16

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

```
package com.udayan.ocp;
import java.io.BufferedReader;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
public class Test {
    public static void main(String[] args)  throws IOException {
        Path path = Paths.get("C:", "TEMP", "msg");

        try (BufferedReader reader =
                    Files.newBufferedReader(path)) {
            String str = null;
            while ((str = reader.readLine()) != null) {
                System.out.println(str);
            }
        }
    }
}
```

16

Options

A. Compilation error.

B. An exception is thrown at runtime.

C. Program executes successfully and produces no output.

D. Program executes successfully and prints 'Welcome!' on to the console.

16

Answer

A. Compilation error.

B. An exception is thrown at runtime.

C. Program executes successfully and produces no output.

[✓ D. Program executes successfully and prints 'Welcome!' on to the console.]

16

Answer

Note that Windows shortcut and symbolic links are different.
Shortcut is just a regular file and symbolic link is a File System object.

To create symbolic link used in this question, I used below command:
C:\TEMP>mklink msg .\Parent\Child\Message.txt

And below message was displayed on to the console for the successful creation of symbolic link 'symbolic link created for msg <<===>> .\Parent\Child\Message.txt'.

Files class has methods such as newInputStream(...), newOutputStream(...), newBufferedReader(...) and newBufferedWriter(...) for files reading and writing.
Give code doesn't cause any compilation error.

path refers to 'C\TEMP\msg', which is a symbolic link and hence Files.newBufferedReader(path) works with 'C:\TEMP\Parent\Child\Message.txt'.

Given code successfully reads the file and prints 'Welcome!' on to the console.

e console.

17

Given structure of EMPLOYEE table:

```
EMPLOYEE (ID integer, FIRSTNAME varchar(100), LASTNAME
varchar(100), SALARY real, PRIMARY KEY (ID))
```

EMPLOYEE table contains below records:

101	John	Smith	12000
102	Sean	Smith	15000
103	Regina	Williams	15500
104	Natasha	George	14600

Also assume:
URL is correct and db credentials are: root/password.
SQL query is correct and valid.
The JDBC 4.2 driver jar is configured in the classpath.

17

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

```java
package com.udayan.ocp;
import java.sql.*;

public class Test {
    public static void main(String[] args)   throws SQLException {
        String url = "jdbc:mysql://localhost:3306/ocp";
        String user = "root";
        String password = "password";
        String query = "Select ID, FIRSTNAME, LASTNAME, " +  "SALARY FROM EMPLOYEE ORDER BY ID";
        try (Connection con = DriverManager.getConnection(url, user, password);
             Statement stmt = con.createStatement(
                     ResultSet.TYPE_SCROLL_INSENSITIVE,
                     ResultSet.CONCUR_READ_ONLY);
        ) {
            ResultSet rs = stmt.executeQuery(query);
            rs.relative(-3);
            rs.relative(1);
            System.out.println(rs.getInt(1));
        }
    }
}
```

17

Options

A. An exception is thrown at runtime.
B. 101
C. 102
D. 103
E. 104

17

Answer

A. An exception is thrown at runtime.

[✓ B. 101]

C. 102

D. 103

E. 104

17

Answer

Given query returns below records:

101	John	Smith	12000
102	Sean	Smith	15000
103	Regina	Williams	15500
104	Natasha	George	14600

Initially cursor is just before the 1st record.

'rs.relative(-3);' doesn't throw any exception but keeps the cursor just before the 1st record. According to javadoc of relative method, "Attempting to move beyond the first/last row in the result set positions the cursor before/after the first/last row". Same is true for absolute method as well.

'rs.relative(1);' is identical to 'rs.next()' so it moves the cursor to the 1st record.

'rs.getInt(1)' returns 101.

18 Assume that all below *.properties files are included in the CLASSPATH. What will be the output of compiling and executing Test class?

//1. ResourceBundle.properties locale=French/Canada	*//2. ResourceBundle_CA.properties* locale=Canada
//3. ResourceBundle_hi.properties locale=Hindi	*//4. ResourceBundle_IN.properties* locale=India

```java
//5. Test.java
package com.udayan.ocp;

import java.util.Locale;
import java.util.ResourceBundle;

public class Test {
    public static void main(String[] args) {
        Locale.setDefault(new Locale("fr", "CA"));
        Locale loc = new Locale("en", "IN");
        ResourceBundle rb = ResourceBundle.getBundle(
                            "ResourceBundle", loc);
        System.out.println(rb.getObject("locale"));
    }
}
```

18

Options

A. French/Canada
B. Canada
C. Hindi
D. India
E. Runtime Exception

18

Answer

[✓ A. French/Canada]

B. Canada

C. Hindi

D. India

E. Runtime Exception

Answer

ResourceBundle.getBundle("ResourceBundle", loc); => Base resource bundle file name should be 'ResourceBundle'.

Default Locale is: fr_CA and passed Locale to getBundle method is: en_IN

The search order for matching resource bundle is:
ResourceBundle_en_IN.properties [1st: Complete en_IN].
ResourceBundle_en.properties [2nd: Only language en].
ResourceBundle_fr_CA.properties [3rd: Complete default Locale fr_CA].
ResourceBundle_fr.properties [4th: Language of default Locale fr].
ResourceBundle.properties [5th: ResourceBundle's name without language or country].

Out of the given resource bundles, 'ResourceBundle.properties' matches.
This resource bundle has key 'locale' and value 'French/Canada'.
rb.getObject("locale") prints 'French/Canada' on to the console.

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

```
package com.udayan.ocp;

import java.util.concurrent.*;

class Printer implements Callable<String> {
    public String call() {
        System.out.println("DONE");
        return null;
    }
}
public class Test {
    public static void main(String[] args) {
        ExecutorService es = Executors.
                    newFixedThreadPool(1);
        es.submit(new Printer());
        System.out.println("HELLO");
        es.shutdown();
    }
}
```

Options

A. HELLO will always be printed before DONE.

B. DONE will always be printed before HELLO.

C. HELLO and DONE will be printed but printing order is not fixed.

D. HELLO will never be printed.

Answer

A. HELLO will always be printed before DONE.

B. DONE will always be printed before HELLO.

[✓ C. HELLO and DONE will be printed but printing order is not fixed.]

D. HELLO will never be printed.

19

Answer

A. HELLO will always be printed before DONE.

B. DONE will always be printed before HELLO.

✓ C. HELLO and DONE will be printed but printing order is not fixed.

D. HELLO will never be printed.

'es.submit(new Printer());' is asynchronous call, hence 2 threads(main and thread from pool) always run in asynchronous mode. Thread execution depends upon underlying OS/JVM's thread scheduling mechanism.

HELLO and DONE will be printed but their order cannot be predicted.

20

Consider below codes:

```
package com.udayan.ocp;
class A<T extends String> {

}
class B<T super String> {
}
```

Which of the following statement is correct?

A.	Both class A and B compiles successfully
B.	Only class A compiles successfully
C.	Only class B compiles successfully

20

Consider below codes:

```
package com.udayan.ocp;
class A<T extends String> {

}
class B<T super String> {
}
```

Which of the following statement is correct?

A.	Both class A and B compiles successfully
[✓ B.]	Only class A compiles successfully
C.	Only class B compiles successfully

20

Answer

A.	Both class A and B compiles successfully
✓ B.	Only class A compiles successfully
C.	Only class B compiles successfully

super is used with wildcard (?) only.

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

```
package com.udayan.ocp;

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

public class Test {
    public static void main(String[] args) {
        List<String> list1 = new ArrayList<>();
        list1.add("A");
        list1.add("B");

        List<? extends Object> list2 = list1;
        list2.remove("A"); //Line 13
        list2.add("C"); //Line 14

        System.out.println(list2);
    }
}
```

21

Options

A. ABC

B. BC

C. Runtime exception

D. Compilation error

21

Answer

A. ABC

B. BC

C. Runtime exception

[✓ D. Compilation error]

Answer

list1 is of List<String> type and contains 2 elements "A" and "B".

list2 is of List<? extends Object> type, which means any List whose type extends from Object. As String extends Object, hence 'List<? extends Object> list2 = list1;' works.

list2.remove("A"); => remove is non-generic method. remove(Object) will be invoked and it will successfully remove "A" from list2.

list2.add("C"); => add is a generic method. add(? extends Object) would be invoked. This means it can take an instance of any UnknownType (extending from Object class).
Compiler can never be sure whether passed argument is a subtype of UnknownType (extending from Object class). Line 14 causes compilation failure.

Remember: Compiler works with reference types and not instances.

22

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

```
package com.udayan.ocp;

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

public class Test {
    public static void main(String[] args) {
        int ref = 10;
        List<Integer> list = new ArrayList<>();
        list.stream().anyMatch(i -> {
            System.out.println("HELLO");
            return i > ref;
        });
    }
}
```

22

Options

A. HELLO

B. Compilation error

C. Program executes successfully but nothing is printed on to the console.

Answer

A. HELLO

B. Compilation error

[✓ C. Program executes successfully but nothing is printed on to the console.]

22

Answer

A. HELLO

B. Compilation error

[✓ C. Program executes successfully but nothing is printed on to the console.]

Method signature for anyMatch method:
boolean anyMatch(Predicate<? super T>) : Returns true if any of the stream element matches the given Predicate. If stream is empty, it returns false and predicate is not evaluated.

As given stream is empty, hence predicate is not evaluated and nothing is printed on to the console.

23

Fill in the blanks:

```
package com.udayan.ocp;

import java.util.concurrent.*;

public class Task extends _________________ {
    @Override
    protected Long compute() {
        return null;
    }
}
```

23

Options

Select All That Apply

A. RecursiveTask<Long>
B. RecursiveTask
C. RecursiveAction
D. RecursiveAction<Long>
E. RecursiveTask<Object>
F. RecursiveAction<Object>

23

Answer

Select All That Apply

- [✓ A. RecursiveTask<Long>]
- [✓ B. RecursiveTask]
- C. RecursiveAction
- D. RecursiveAction<Long>
- [✓ E. RecursiveTask<Object>]
- F. RecursiveAction<Object>

23

Answer

RecursiveTask is a generic class which extends from ForkJoinTask<V> class.
RecursiveTask<V> declares compute() method as: 'protected abstract V compute();'
In the given code overriding method returns Long, so classes from which class Task can extend are: RecursiveTask<Long>, RecursiveTask<Object> [co-variant return type in overriding method] and RecursiveTask [co-variant return type in overriding method].

RecursiveAction is a non-generic class which extends from ForkJoinTask<Void> class.
RecursiveAction declared compute() method as: 'protected abstract void compute();'
In the given code overriding method returns Long, hence RecursiveAction can't be used as super class of Task.

24

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

```
package com.udayan.ocp;

import java.util.Scanner;

public class Test {
    public static void main(String[] args) {
        try(Scanner scan = new Scanner(System.in)) {
            String s = scan.nextLine();
            System.out.println(s);
            scan = null;
        }
    }
}
```

24

Options

A. Normal Termination

B. Exception is thrown at runtime

C. Compilation error

24

Answer

A. Normal Termination

B. Exception is thrown at runtime

[✓ C. Compilation error]

24

Answer

A. Normal Termination

B. Exception is thrown at runtime

[✓ C. Compilation error]

Resources used in try-with-resources statement are implicitly final, which means they can't be reassigned. scan = null; will fail to compile as we are trying to assign null to variable scan.

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

```
package com.udayan.ocp;

import
java.util.concurrent.ExecutionException;
import
java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

public class Test {
  private static void print() {
    System.out.println("PRINT");
  }
```

```
  private static Integer get() {
    return 10;
  }
  public static void main(String [] args)
      throws InterruptedException,
ExecutionException {
    ExecutorService es =

Executors.newFixedThreadPool(10);
    Future<?> future1 =
es.submit(Test::print);
    Future<?> future2 =
es.submit(Test::get);
    System.out.println(future1.get());
    System.out.println(future2.get());
    es.shutdown();
  }
}
```

25

Options

A.	Compilation error	**B.**	PRINT null 10
C.	null 10 PRINT	**D.**	PRINT 10 Null
E	null PRINT 10		

25

Answer

A.	Compilation error	✓ B.	PRINT null 10
C.	null 10 PRINT	D.	PRINT 10 Null
E	null PRINT 10		

25

Answer

A.	Compilation error	[✓ B.]	PRINT null 10
C.	null	D.	PRINT

Method reference 'Test::print' is for the run() method implementation of Runnable and 'Test::get' is for the call() method implementation of Callable.
Future<?> is valid return type for both the method calls. get() method throws 2 checked exceptions: InterruptedException and ExecutionException, hence given code compiles fine.
get() method waits for task completion, hence 'PRINT' will be printed first.
future1.get() returns null and future2.get() returns 10.

OCA — 1Z0-809
Exam Practice Test