Wrapper

Integer -> Double is not possible as they all belong to Number Class. Siblings

Interger-> int (AutoUnboxing) – double (implicit type cast)

Question 9: **Incorrect**

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

```
1  package com.udayan.oca;
2
3  public class Test {
4
5      private static void add(double d1, double d2) {
6          System.out.println("double version: " + (d1 + d2));
7      }
8
9      private static void add(Double d1, Double d2) {
10         System.out.println("Double version: " + (d1 + d2));
11     }
12
13     public static void main(String[] args) {
14         add(10.0, new Integer(10));
15     }
16
17 }
```

- ◯ **An exception is thrown at runtime**
- ◯ **Double version: 20.0**
- ◉ **Compilation error** **(Incorrect)**
- ◯ **double version: 20.0** **(Correct)**

Explanation

int can be converted to double but Integer type can't be converted to Double type as Integer and Double are siblings (both extends from Number class) so can't be casted to each other.

add(10.0, new Integer(10)); => 1st parameter is tagged to double primitive type and 2nd parameter is converted to int, is tagged to double primitive type as well. So, add(double, double); method is invoked.

Two instance of following wrapper objects

Boolean, Byte ,

Character from (\u0000 to \u007f (7f equals to 127))

Short and Integer from -128 to 127. (Integer and short between value of -128 to 127, byte range) , created through autoboxing will always be same if their primitive values are same:

```
Integer d2 = 127;
Integer d3 = 127;
System.out.println(d2==d3); - True
```

```
Boolean b = true;
Boolean a = true;
System.out.println(a==b);
```

```
Boolean b = null;
System.out.println(b); -> null
System.out.println(b.toString()); -> Null Pointer Exception
```

```
Boolean b = null;
if(b){                  -> Unboxing of b produces null pointer exception
    System.out.println("1");
}
```

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.

Question 14: **Incorrect**

Below is the code of Test.java file:

```
1  package com.udayan.oca;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  public class Test {
7       public static void main(String [] args) {
8           List<Integer> list = new ArrayList<Integer>();
9
10          list.add(27);
11          list.add(27);
12
13          list.add(new Integer(27));
14          list.add(new Integer(27));
15
16          System.out.println(list.get(0) == list.get(1));
17          System.out.println(list.get(2) == list.get(3));
18      }
19 }
```

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

- ○ **false**
  true
- ○ **true**
  true
- ○ **true**
  false **(Correct)**
- ◉ **false**
  false **(Incorrect)**

Question 21: **Incorrect**

Which of the following are Java Exception classes?

Select 3 options.

- [ ] **NullException**
- [ ] **NumberFormatException** (Correct)
- [x] **IllegalArgumentException** (Correct)
- [ ] **ArrayIndexException**
- [x] **ClassCastException** (Correct)

Explanation

ClassCastException, NumberFormatException and IllegalArgumentException are Runtime exceptions.

There are no exception classes in java with the names: NullException and ArrayIndexException.

Explanation
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: [100, 200, 100, 200]. list.remove(new Integer(100)); removes the first occurrence of 100 from the list, which means the 1st element of the list.

After removal list contains: [200, 100, 200].

NOTE: String class and all the wrapper classes override equals(Object) method, hence at the time of removal when another instance is passes[new Integer(100)], there is no issue in removing the matching item.

```
1  package com.udayan.oca;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  public class Test {
7       public static void main(String[] args) {
8           List<Integer> list = new ArrayList<>();
9           list.add(100);
10          list.add(200);
11          list.add(100);
12          list.add(200);
13          list.remove(new Integer(100));
14
15          System.out.println(list);
16      }
17 }
```

- ◯ **Compilation error**
- ◯ **[200]**
- ◯ **[100, 200, 200]**
- ◉ **Exception is thrown at runtime** (Incorrect)
- ◯ **[200, 200]**
- ◯ **[200, 100, 200]** (Correct)

```
List<> list = new ArrayList(); -        CE - Java Illegal start of type
List<> list = new ArrayList<>();        CE Illegal start of Type for List<>
List<> list = new ArrayList<String>(); CE Illegal start of Type for List<>

List list = new ArrayList<String>();    Valid (Warning) - Raw Use
List list = new ArrayList<>();          Valid (Warning) - Raw Use
List list = new ArrayList();            Valid (Warning) - Raw Use


List<String> list = new ArrayList();        Valid (Warning) - Raw Use
List<String> list = new ArrayList<>();      Valid No Warning
List<String> list = new ArrayList<String>(); Valid No Warning
```

Type parameter can be ignored from right side but <> need to there to avoid warning

Question 23: **Incorrect**

Consider the code snippet:

```
1  import java.util.ArrayList;
2  import java.util.List;
3
4  public class Test {
5       List list1 = new ArrayList<String>(); //Line 5
6       List<String> list2 = new ArrayList(); //Line 6
7       List<> list3 = new ArrayList<String>(); //Line 7
8       List<String> list4 = new ArrayList<String>(); //Line 8
9       List<String> list5 = new ArrayList<>(); //Line 9
10 }
```

Which of the following statements compile without any warning?

Select 2 options.

Explanation
Line 8's syntax was added in JDK 5 and it compiles without any warnings.

Line 9's syntax was added in JDK 7, in which type parameter can be ignored from right side of the statement, it is inferred from left side, so Line 9 also compiles without any warning.

Type parameter can't be removed from declaration part, hence Line 7 gives compilation error.

Both Line 5 and Line 6 are mixing Generic type with Raw type and hence warning is given by the compiler.

Question 25: **Incorrect**

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

```
1  package com.udayan.oca;
2
3  public class Test {
4      public static void main(String[] args) {
5          StringBuilder sb = new StringBuilder("Hurrah! I Passed...");
6          sb.delete(0, 100);
7          System.out.println(sb.length());
8      }
9  }
```

- ○ **19**
- ○ **0** **(Correct)**
- ○ **16**
- ◉ **StringIndexOutOfBoundsException is thrown at runtime** **(Incorrect)**

Explanation

'delete' method accepts 2 parameters: delete(int start, int end), where start is inclusive and end is exclusive.

This method throws StringIndexOutOfBoundsException for following scenarios:

A. start is negative

B. start is greater than sb.length()

C. start is greater than end

If end is greater than the length of StringBuilder object, then StringIndexOutOfBoundsException is not thrown and end is set to sb.length().

So, in this case, `sb.delete(0, 100);` is equivalent to `sb.delete(0, sb.length());` and this deletes all the characters from the StringBuilder object.

Hence, System.out.println(sb.length()); prints 0 on to the console.

Question 26: **Incorrect**

For the class Test, which options, if used to replace /*INSERT*/, will print TEN on to the console? Select 4 options.

```
1  package com.udayan.oca;
2
3  public class Test {
4      public static void main(String[] args) {
5          /*INSERT*/
6          switch(var) {
7              case 10:
8                  System.out.println("TEN");
9                  break;
10             default:
11                 System.out.println("DEFAULT");
12         }
13     }
14 }
```

- [x] `byte var = 10;` **(Correct)**
- [x] `Integer var = 10;` **(Correct)**
- [ ] `long var = 10;`
- [x] `Short var = 10;` **(Correct)**
- [ ] `double var = 10;`
- [ ] `char var = 10;` **(Correct)**

Explanation

switch can accept primitive types: byte, short, int, char; wrapper types: Byte, Short, Integer, Character; String and enums.

In this case long and double are invalid values to be passed in switch expression. char uses 16 bits (2 Bytes) and its range is 0 to 65535 (no signed bit reserved) so it can easily store value 10.

Explanation
Method m1 is overloaded to accept 3 different parameters: String, CharSequence and Object.

String implements CharSequence and Object is the super Parent class in Java. There is no conflict among the overloaded methods for the call m1(null) as it is mapped to the class lowest in hierarchy, which is String class. Hence, output will be "String".

Now if you add one more overloaded method, `static void m1(StringBuilder s) {...}` in the Test class, then `m1(null);` would cause compilation error as it would match to both m1(StringBuilder) and m1(String) methods. So m1(null) in that case would be ambiguous call and would cause compilation error.

For the same reason, System.out.println(null); causes compilation error as println method is overloaded to accept 3 reference types Object, String and char [] along with primitive types.

System.out.println(null); matches to both println(char[]) and println(String), so it is an ambiguous call and hence the compilation error.

Question 31: **Incorrect**

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

```
1  package com.udayan.oca;
2
3  public class Test {
4      public static void main(String[] args) {
5          m1(null);
6      }
7
8      static void m1(CharSequence s) {
9          System.out.println("CharSequence");
10     }
11
12     static void m1(String s) {
13         System.out.println("String");
14     }
15
16     static void m1(Object s) {
17         System.out.println("Object");
18     }
19 }
```

- ○ **String** (Correct)
- ○ **CharSequence**
- ◉ **Compilation Error** (Incorrect)
- ○ **Object**

Double d = null;
System.*out*.println(d+""); - No CE or RE
System.*out*.println(d+12); this would perform unboxing and null.doubleValue() – RunTime Exception

Question 33: **Incorrect**

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

```
1  package com.udayan.oca;
2
3  public class Test {
4
5      private static void add(double d1, double d2) {
6          System.out.println("double version: " + (d1 + d2));
7      }
8
9      private static void add(Double d1, Double d2) {
10         System.out.println("Double version: " + (d1 + d2));
11     }
12
13     public static void main(String[] args) {
14         add(10.0, null);
15     }
16
17 }
```

- (●) **Double version: 10.0** (Incorrect)
- ( ) **double version: 10.0**
- ( ) **Compilation error**
- ( ) **An exception is thrown at runtime** (Correct)

Explanation

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.

Question 36: **Incorrect**

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.util.ArrayList;
5  import java.util.List;
6
7  public class Test {
8       public static void main(String[] args) {
9           List<String> list = new ArrayList<>();
10          list.add("ONE");
11          list.add("TWO");
12          list.add("THREE");
13          list.add("THREE");
14
15          if(list.remove(2)) {
16              list.remove("THREE");
17          }
18
19          System.out.println(list);
20      }
21 }
```

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

- ◯ **An exception is thrown at runtime**
- ◯ **[ONE, TWO, THREE]**
- ◉ **[ONE, TWO]** **(Incorrect)**
- ◯ **[ONE, TWO, THREE, THREE]**
- ◯ **Compilation error** **(Correct)**

Explanation

list.remove(Object) method returns boolean result but list.remove(int index) returns the removed item from the list, which in this case is of String type and not Boolean type and hence if(list.remove(2)) causes compilation error.

List.remove(0) -> returns the object -> null

List.remove(null)-> returns true if the object is found

Question 39: **Incorrect**

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

```
1  package com.udayan.oca;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  public class Test {
7      public static void main(String[] args) {
8          List<String> list = new ArrayList<>();
9          list.add(null);
10         list.add(null);
11         list.add(null);
12         System.out.println(list.remove(0) + ":" + list.remove(null));
13     }
14 }
```

- ○ **true:false**
- ○ **null:true** (Correct)
- ◉ **true:true** (Incorrect)
- ○ **null:null**
- ○ **NullPointerException is thrown at runtime**

Explanation

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.

```
5  public class Test {
6      public static void main(String[] args) {
7          String [] arr = {"*", "**", "***", "****", "*****"};
8          Predicate pr1 = s -> s.length() < 4;
9          print(arr, pr1);
10     }
11
12     private static void print(String [] arr, Predicate<String> predicate) {
13         for(String str : arr) {
14             if(predicate.test(str)) {
15                 System.out.println(str);
16             }
17         }
18     }
19 }
```

- ○ **Compilation error** **(Correct)**

- ○
  *
  **
  ***
  ****

- ◉
  *
  **
  ***
  (Incorrect)

Explanation
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

InCorrectly Implements

Question 48: **Incorrect**

Consider the following interface declaration:

```
1  public interface I1 {
2      void m1() throws java.io.IOException;
3  }
```

Which of the following incorrectly implements interface I1?

- (●)
```
1  public class C3 implements I1 {
2      public void m1() throws java.io.IOException{}
3  }
```
(Incorrect)

- ( )
```
1  public class C2 implements I1 {
2      public void m1() throws java.io.FileNotFoundException{}
3  }
```

- ( )
```
1  public class C1 implements I1 {
2      public void m1() {}
3  }
```

- ( )
```
1  public class C4 implements I1 {
2      public void m1() throws Exception{}
3  }
```
(Correct)

- **If the superclass method does not declare an exception**
    - If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but it can declare unchecked exception.
- **If the superclass method declares an exception**
    - If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.

When a new String object is created using new Keyword – 2 Objects gets created – Heap Area and SCP.

Compile Time constants -> Literals or Final Variable

Compile time string computation using concatenation will be referred is SCP

RunTime String Computation using concatenation will be referred are newly created. Therfore distinct (If the resultant Expression is not a constant Expression)

Explanation

Please note that Strings computed by concatenation at compile time, will be referred by String Pool during execution. Compile time String concatenation happens when both of the operands are compile time constants, such as literal, final variable etc.

Whereas, Strings computed by concatenation at run time (if the resultant expression is not constant expression) are newly created and therefore distinct.

fName is a constant variable and lName is a non-constant variable.

`fName + lName` is not a constant expression and hence the expression will be computed at run-time and the resultant String object "JamesGosling" will not be referred by String Pool.

As fName is constant variable and "Gosling" is String literal, hence the expression `fName + "Gosling"` is a constant expression, therefore expression is computed at compile-time and results in String literal "JamesGosling".

So, during compilation, Java compiler translates the statement

```
String name2 = fName + "Gosling";
```

to

```
String name2 = "JamesGosling";
```

As "JamesGosling" is a String literal, hence at runtime it will be referred by String Pool.

So, at runtime name1 and name2 refer to different String object and that is why name1 == name2 returns false.

`"James" + "Gosling"` is also a constant expression and hence Java compiler translates the statement

```
String name3 = "James" + "Gosling";
```

to

```
String name3 = "JamesGosling";
```

This means at runtime, variable 'name3' will refer to the same String pool object "JamesGosling", which is referred by variable 'name3'.

So, name2 and name3 refer to same String object and that is why name2 == name3 returns true.

Fname – Constant

Lname – non constant

Name1 = fname+lname -> Non Constant -> Run Time NewObject

Name 2-> fname+ “Cosling” -> Final Constant + Literal Constant -> String Pool (JamesGosling)

Name 3 -> String Pool (JamesGosling)

False

True

Question 56: **Incorrect**

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  public class Test {
5      public static void main(String[] args) {
6          final String fName = "James";
7          String lName = "Gosling";
8          String name1 = fName + lName;
9          String name2 = fName + "Gosling";
10         String name3 = "James" + "Gosling";
11         System.out.println(name1 == name2);
12         System.out.println(name2 == name3);
13     }
14 }
```

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

- (●) **true**
  true **(Incorrect)**
- ( ) **false**
  false
- ( ) **false**
  true **(Correct)**
- ( ) **true**
  false

Only long float and double can be suffixed by l f d

```
1  package com.udayan.oca;
2
3  public class Test {
4      public static void main(String[] args) {
5          char c = 'Z';
6          long l = 100_001;
7          int i = 9_2;
8          float f = 2.02f;
9          double d = 10_0.35d;
10         l = c + i;
11         f = c * l * i * f;
12         f = l + i + c;
13         i = (int)d;
14         f = (long)d;
15     }
16 }
```

Does above code compile successfully?

- ○ **Yes** **(Correct)**
- ◉ **No** **(Incorrect)**

Explanation

For readability purpose underscore (_) is used to separate numeric values. This is very useful in representing big numbers such as credit card numbers (1234_7654_9876_0987). long data can be suffixed by l, float by f and double by d. So first 5 variable declaration and assignment statements inside main(String []) method don't cause any compilation error.

Let's check rest of the statements:

l = c + i; => Left side variable 'l' is of long type and right side expression evaluates to an int value, which can easily be assigned to long type. No compilation error here.

f = c * l * i * f; => Left side variable 'f' is of float type and right side expression evaluates to float value, which can easily be assigned to float type. Hence, it compiles successfully.

f = l + i + c; => Left side variable 'f' is of float type and right side expression evaluates to long value, which can easily be assigned to float type. Hence, no issues here.

i = (int)d; => double can't be assigned to int without explicit casting, right side expression `(int)d;` is casting double to int, so no issues.

f = (long)d; => double can't be assigned to float without explicit casting, right side expression `(long)d;` is casting double to long, which can easily be assigned to float type. It compiles successfully.

Question 60: **Incorrect**

What is the output if below program is run with the command line:

java Test

```
1  public class Test {
2      public static void main(String[] args) {
3          System.out.println(args.length);
4      }
5  }
```

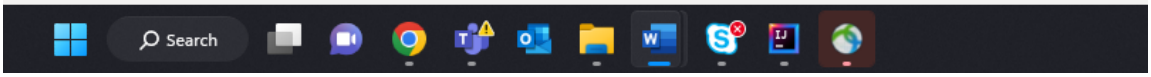
- ○ **ArrayIndexOutOfBoundsException**
- ◉ **NullPointerException** **(Incorrect)**
- ○ **1**
- ○ **0** **(Correct)**

Explanation

We have not passed any command-line arguments, hence args refers to an array object of Size 0.

args.length prints 0. args is not null and hence no NullPointerException.

Also we are not accessing array element so no question of



All wrapper classes are immutable. Any operations would create a new object and not modify the old

Question 66: **Incorrect**

Consider below code:

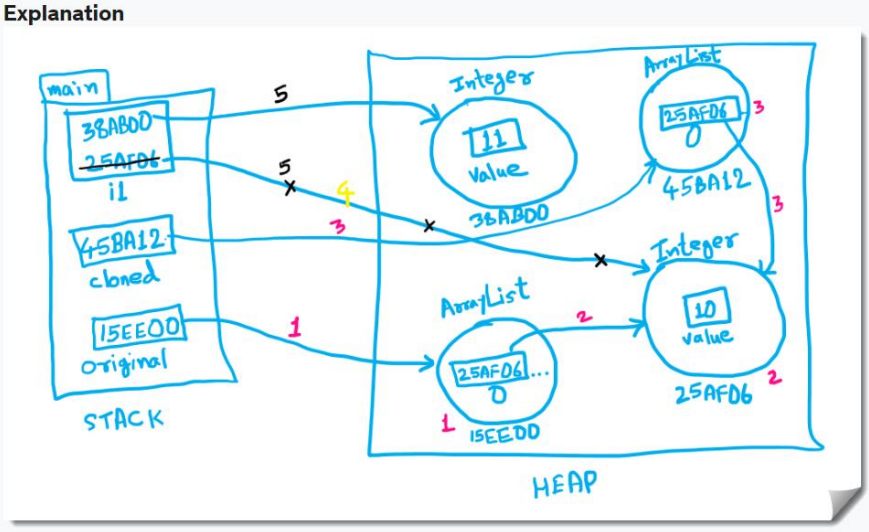
```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.util.ArrayList;
5
6  public class Test {
7      public static void main(String[] args) {
8          ArrayList<Integer> original = new ArrayList<>();
9          original.add(new Integer(10));
10
11         ArrayList<Integer> cloned = (ArrayList<Integer>) original.clone();
12         Integer i1 = cloned.get(0);
13         ++i1;
14
15         System.out.println(cloned);
16     }
17 }
```

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

- ○ **[10]** (Correct)
- ○ **[11]**
- ◉ **An exception is thrown at runtime** (Incorrect)
- ○ **Compilation error**

Explanation

Explanation



5. `++i1;` => As Integer object is immutable, hence ++i1; creates a new Integer object with value 11 and suppose this newly created Integer object is stored at memory location 38AB00. This means variable 'i1' stops referring to Integer object at the memory location 25AF06 and starts referring to Integer object at the memory location 38AB00.

Cloned list stays intact and still refers to Integer object at memory location 25AF06.

M – Month

m- minute

D- Day of the year

d- day of the month

```
7
8  public class Test {
9      public static void main(String [] args) {
10         LocalDate date = LocalDate.of(2012, 1, 11);
11         Period period = Period.ofMonths(2);
12         DateTimeFormatter formatter = DateTimeFormatter.ofPattern("mm-dd-yy");
13         System.out.print(formatter.format(date.minus(period)));
14     }
15 }
```

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

- ◯ **11-11-12**
- ◯ **01-11-11**
- ◯ **11-11-11**
- ◯ **01-11-12**
- ◯ **Runtime exception** (Correct)

Explanation

While working with dates, programmers get confused with M & m and D & d.

Easy way to remember is that Bigger(Upper case) letters represent something bigger. M represents month & m represents minute, D represents day of the year & d represents day of the month.

LocalDate's object doesn't have time component, mm represents minute and not months so at runtime format method throws exception.

```java
package com.udayan.oca;

import java.time.LocalDate;

public class Test {
     public static void main(String [] args) {
         LocalDate date = LocalDate.parse("1980-03-16");
         System.out.println(date.minusYears(-5));
     }
}
```

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

- ◯ **Compilation error**
- ◯ **Runtime exception**
- ◯ **1985-03-16** **(Correct)**
- ◯ **1975-03-16**

Explanation

minusYears, minusMonths, minusWeeks, minusDays methods accept long parameter so you can pass either positive or negative value.

If positive value is passed, then that specified value is subtracted and if negative value is passed, then that specified value is added. I think you still remember: minus minus is plus.

Similarly plusYears, plusMonths, plusWeeks, plusDays methods work in the same manner.

If positive value is passed, then that specified value is added and if negative value is passed, then that specified value is subtracted.

Question 30: **Skipped**

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.time.LocalDate;
5  import java.time.Period;
6
7  public class Test {
8       public static void main(String [] args) {
9           LocalDate date = LocalDate.parse("2000-01-01");
10          Period period = Period.ofYears(-3000);
11          System.out.println(date.plus(period));
12      }
13 }
```

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

- ◯ **5000-01-01**
- ◯ **Compilation error**
- ◯ **1000-01-01**
- ◯ **Runtime exception**
- ◯ **-1000-01-01** **(Correct)**

Explanation

The minimum supported LocalDate is: {-999999999-01-01} and maximum supported LocalDate is: {+999999999-12-31}.

If period of -3000 years is added to 1st Jan 2000, then result is 1st Jan -1000.

```
8  public class Test {
9      public static void main(String[] args) {
10         List<String> dryFruits = new ArrayList<>();
11         dryFruits.add("Walnut");
12         dryFruits.add("Apricot");
13         dryFruits.add("Almond");
14         dryFruits.add("Date");
15
16         ListIterator<String> iterator = dryFruits.listIterator();
17         while(iterator.hasNext()) {
18             if(iterator.next().startsWith("A")) {
19                 iterator.remove();
20              }
21         }
22
23         System.out.println(dryFruits);
24     }
25 }
```

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

- ◯ **Compilation error**
- ◯ **An exception is thrown at runtime**
- ◯ **[Walnut, Date]** **(Correct)**
- ◯ **[Walnut, Apricot, Almond, Date]**

Explanation

If you want to remove the items from ArrayList, while using Iterator or ListIterator, then use Iterator.remove() or ListIterator.remove() method and NOT List.remove() method.

In this case ListIterator.remove() method is used. startsWith("A") returns true for "Apricot" and "Almond" so these elements are removed from the list. In the output, [Walnut, Date] is displayed.

```
7
8  public class Test {
9      public static void main(String[] args) {
10         List<LocalDate> dates = new ArrayList<>();
11         dates.add(LocalDate.parse("2018-7-11"));
12         dates.add(LocalDate.parse("1919-10-25"));
13         dates.add(LocalDate.of(2020, 4, 8));
14         dates.add(LocalDate.of(1980, Month.DECEMBER, 31));
15
16         dates.removeIf(x -> x.getYear() < 2000);
17
18         System.out.println(dates);
19     }
20 }
```

- ◯ **[1919-02-25, 1980-12-31]**
- ◯ **Runtime exception** **(Correct)**
- ◯ **[2018-07-11, 2020-04-08]**
- ◯ **[2018-07-11, 1919-02-25, 2020-04-08, 1980-12-31]**

Explanation

LocalDate.parse(CharSequence text) method accepts String in "9999-99-99" format only, in which month and day part in the passed object referred by text should be of 2 digits, such as to represent MARCH, use 03 and not 3 & to represent 4th day of the month, use 04 and not 4.

Single digit month and day value are not automatically padded with 0 to convert it to 2 digits.

To represent 9th June 2018, format String must be "2018-06-09".

If you pass "2018-6-9" or "2018-06-9" or "2018-6-09" (not in correct formats), then an instance of java.time.format.DateTimeParseException will be thrown.

In this question, LocalDate.parse("2018-7-11") throws an exception at runtime as JULY is represented as 7, whereas it should be represented as 07.

Question 46: **Skipped**

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.time.LocalDate;
5
6  public class Test {
7       public static void main(String [] args) {
8           LocalDate date = LocalDate.parse("2018-1-01");
9           System.out.println(date);
10      }
11 }
```

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

- ◯ **An exception is thrown at runtime** (Correct)
- ◯ **2018-1-01**
- ◯ **2018-1-1**
- ◯ **2018-01-01**

Explanation

LocalDate.parse(CharSequence) method accepts String in "9999-99-99" format only. Single digit month and day value are padded with 0 to convert it to 2 digits.

To represent 9th June 2018, format String must be "2018-06-09".

If correct format string is not passed then an instance of java.time.format.DateTimeParseException is thrown.

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

```
1  package com.udayan.oca;
2
3  public class Test {
4       public static void main(String [] args) {
5           int a = 2;
6           boolean res = false;
7           res = a++ == 2 || --a == 2 && --a == 2;
8           System.out.println(a);
9       }
10 }
```

- ○ **2**
- ○ **Compilation error**
- ○ **1**
- ○ **3** **(Correct)**

Explanation

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.

Question 59: **Skipped**

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.time.Period;
5
6  public class Test {
7       public static void main(String [] args) {
8           Period period = Period.of(2, 1, 0).ofYears(10).ofMonths(5).ofDays(2);
9           System.out.println(period);
10      }
11 }
```

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

- ○ **P12Y6M2D**
- ○ **P2Y1M0D**
- ○ **P2Y1M**
- ○ **P2D** **(Correct)**

Explanation

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.

Question 67: Skipped

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.time.LocalDate;
5
6  public class Test {
7       public static void main(String [] args) {
8           LocalDate d1 = LocalDate.parse("1999-09-09");
9           LocalDate d2 = LocalDate.parse("1999-09-09");
10          LocalDate d3 = LocalDate.of(1999, 9, 9);
11          LocalDate d4 = LocalDate.of(1999, 9, 9);
12          System.out.println((d1 == d2) + ":" + (d2 == d3) + ":" + (d3 == d4));
13      }
14 }
```

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

- ○ **false:false:true**
- ○ **true:true:true**
- ○ **false:false:false** (Correct)
- ○ **true:false:true**

Explanation

"parse" and "of" methods create new instances, so in this case you get 4 different instance of LocalDate stored at 4 different memory addresses.

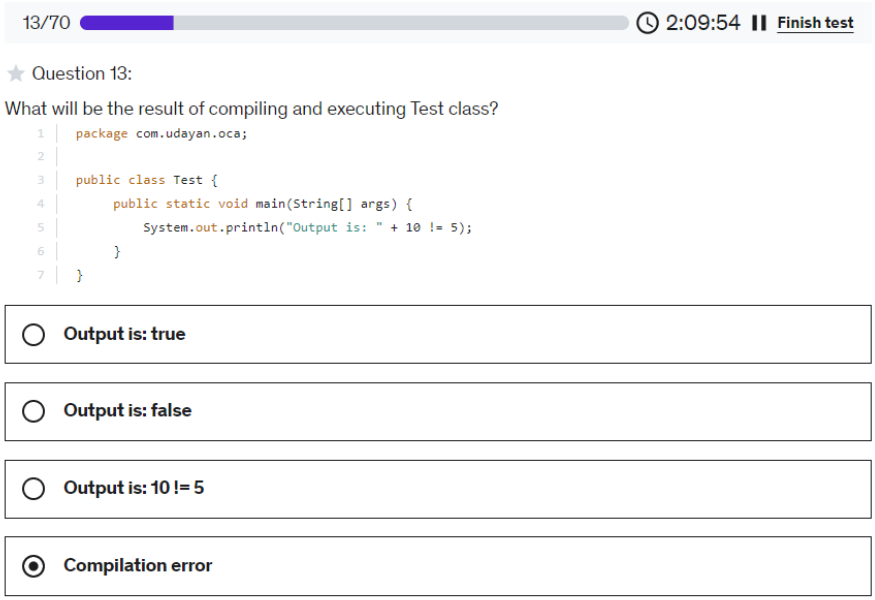
StringBuilder sb= New StringBuilder(null) -. Null pointer Exception

New StringBuilder.append(null) -> null value ambiguous between StringBuilder.append(String) or (CharSequence) or (Object) or (StringBuilder) or (Char[])

String str = null;

Sop(str) -> null

Sop(null) -> CE ambiguous between String and Char[]



```java
class Test {
    public static void main(String[] args) {


        String str1 = new String( original: "Core");
        String str2 = new String( original: "CoRe");
        System.out.println(str1=str2);// ->CoRe

    int i=5;
    int j=10;
    System.out.println(i=j);

    }
}
```

18/70 2:08:13 Finish test

Question 18:

Consider below code:

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

import java.time.Period;

public class Test {
     public static void main(String [] args) {
         Period period = Period.of(0, 0, 0);
         System.out.println(period);
     }
}
```

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

- (●) **P0D**
- ( ) **p0d**
- ( ) **p0y0m0d**
- ( ) **P0Y0M0D**

20/70 2:06:13 Finish test

Question 20:

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.time.LocalDate;
5  import java.time.LocalTime;
6
7  public class Test {
8       public static void main(String [] args) {
9           LocalDate date = LocalDate.parse("1947-08-14");
10          LocalTime time = LocalTime.MAX;
11          System.out.println(date.atTime(time));
12      }
13 }
```

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

- ◯ **1947-08-14T23:59:59.999999999**
- ◯ **1947-08-14T23:59:59.999**
- ◯ **1947-08-14T23:59:59.0**
- ◯ **1947-08-14T23:59:59**

Question 21:

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

```
1  package com.udayan.oca;
2
3  public class Test {
4       public static void main(String[] args) {
5           double [] arr = new int[2]; //Line 3
6           System.out.println(arr[0]); //Line 4
7       }
8  }
```

- ○ **0.0**
- ◉ **Line 3 causes compilation error**
- ○ **0**
- ○ **Line 4 causes runtime exception**

Given code:

```
1  package com.udayan.oca;
2
3  public class Test {
4      public static void main(String[] args) {
5          String [] arr = {"I", "N", "S", "E", "R", "T"};
6          for(/*INSERT*/) {
7              if (n % 2 == 0) {
8                  continue;
9              }
10             System.out.print(arr[n]); //Line n1
11         }
12     }
13 }
```

And below options:

1. int n = 0; n < arr.length; n += 1
2. int n = 0; n <= arr.length; n += 1
3. int n = 1; n < arr.length; n += 2
4. int n = 1; n <= arr.length; n += 2

How many above options can be used to replace /*INSERT*/, such that on execution, code will print NET on to the console?

- ○ **Only one option**
- ○ **Only three options**
- ◉ **Only two options**
- ○ **None of the other options**
- ○ **All four options**

Question 28:

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.time.LocalDate;
5
6  class MyLocalDate extends LocalDate {
7       @Override
8       public String toString() {
9           return super.getDayOfMonth() + "-" + super.getMonthValue() +
10             "-" +  super.getYear();
11      }
12 }
13
14 public class Test {
15      public static void main(String [] args) {
16          MyLocalDate date = LocalDate.parse("1980-03-16");
17          System.out.println(date);
18      }
19 }
```

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

- ○ **An exception is thrown at runtime**
- ○ **16-03-1980**
- ○ **16-3-1980**
- ○ **1980-03-16**
- ◉ **Compilation error**

Question 29:

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

```
1  package com.udayan.oca;
2
3  public class Test {
4       public static void main(String[] args) {
5           int grade = 60;
6           if(grade = 60)
7               System.out.println("You passed...");
8           else
9               System.out.println("You failed...");
10      }
11 }
```

- ○ **You passed...**
- ○ **You failed...**
- ○ **Produces no output**
- ◉ **Compilation error**

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

public class A {
     public int i1 = 1;
     protected int i2 = 2;
}
```

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

import com.udayan.oca.A;

public class B extends A {
     public void print() {
         A obj = new A();
         System.out.println(obj.i1); //Line 8
         System.out.println(obj.i2); //Line 9
         System.out.println(this.i2); //Line 10
         System.out.println(super.i2); //Line 11
     }

     public static void main(String [] args) {
         new B().print();
     }
}
```

One of the statements inside print() method is causing compilation failure. Which of the below solutions will help to resolve compilation error?

- ○ **Comment the statement at Line 10**
- ◉ **Comment the statement at Line 8**
- ○ **Comment the statement at Line 9**
- ○ **Comment the statement at Line 11**

Question 33:

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

```
1  package com.udayan.oca;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  public class Test {
7      public static void main(String[] args) {
8          String[] names = { "Smith", "Brown", "Thomas", "Taylor", "Jones" };
9          List<String> list = new ArrayList<>();
10         for (int x = 0; x < names.length; x++) {
11             list.add(names[x]);
12             switch (x) {
13                 case 2:
14                     continue;
15             }
16             break;
17         }
18         System.out.println(list.size());
19     }
20 }
```

- ○ 2
- ○ 3
- ○ 5
- ○ 4
- ○ 0
- ○ None of the other options
- ◉ 1

Question 35:

Consider code of Test.java file:

```
1  package com.udayan.oca;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  public class Test {
7      public static void main(String[] args) {
8          List<Character> list = new ArrayList<>();
9          list.add(0, 'V');
10         list.add('T');
11         list.add(1, 'E');
12         list.add(3, 'O');
13
14         if(list.contains('O')) {
15             list.remove('O');
16         }
17
18         for(char ch : list) {
19             System.out.print(ch);
20         }
21     }
22 }
```

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

- (●) **VET**
- ( ) **VTEO**
- ( ) **Compilation error**
- ( ) **VTE**
- ( ) **VETO**
- ( ) **Runtime exception**

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

```
1  package com.udayan.oca;
2
3  public class Test {
4       public static void main(String[] args) {
5           try {
6               main(args);
7           } catch (Exception ex) {
8               System.out.println("CATCH-");
9           }
10              System.out.println("OUT");
11      }
12 }
```

- (●) **None of the System.out.println statements are executed**
- ( ) **Compilation error**
- ( ) **CATCH-OUT**
- ( ) **OUT**

[2,1]

38/70 1:39:56 Finish test

★ Question 38:

Below is the code of Test.java file:

```
1  package com.udayan.oca;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  public class Test {
7       public static void main(String [] args) {
8           List<Integer> list = new ArrayList<Integer>();
9           list.add(new Integer(2));
10          list.add(new Integer(1));
11          list.add(new Integer(0));
12
13          list.remove(list.indexOf(0));
14
15          System.out.println(list);
16      }
17 }
```

Question 41:

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  public class Test {
5      public static void main(String[] args) {
6          String s1 = "OCAJP";
7          String s2 = "OCAJP" + "";
8          System.out.println(s1 == s2);
9      }
10 }
```

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

- ○ **OCAJP**
- ○ **Compilation error**
- ○ **false**
- ◉ **true**

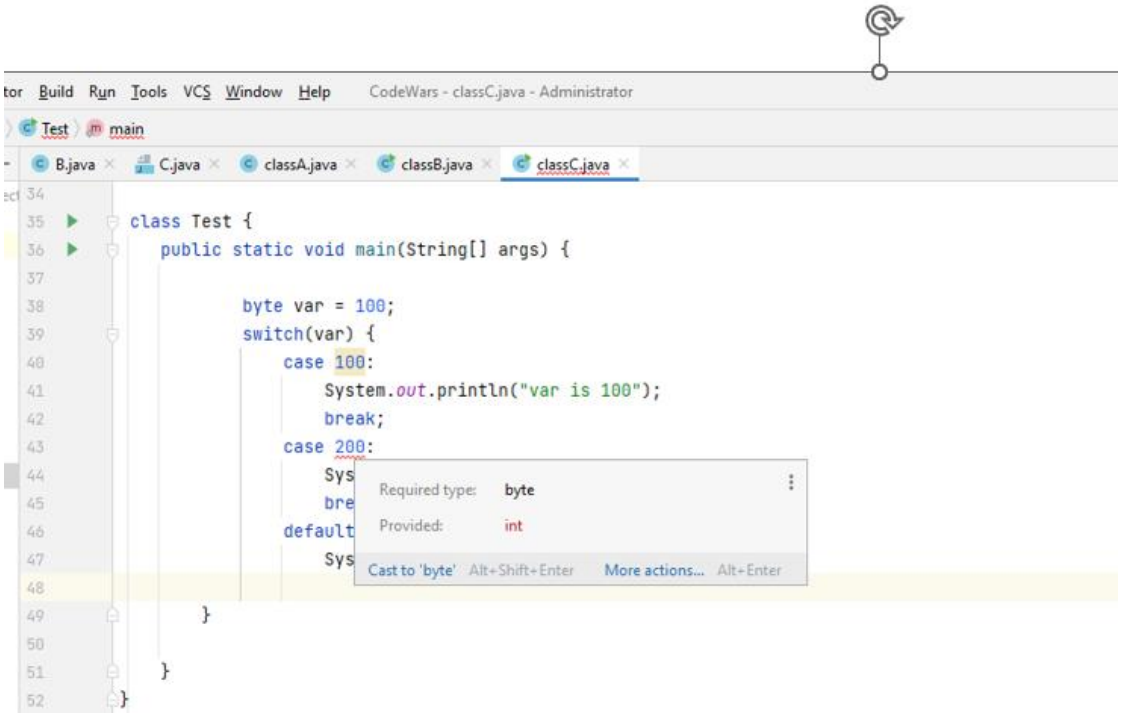
```
package com.udayan.oca;

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();
     }
}
```

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

- (●) **ClassCastException is thrown at runtime**
- ( ) **A**
- ( ) **Compilation error**
- ( ) **B**

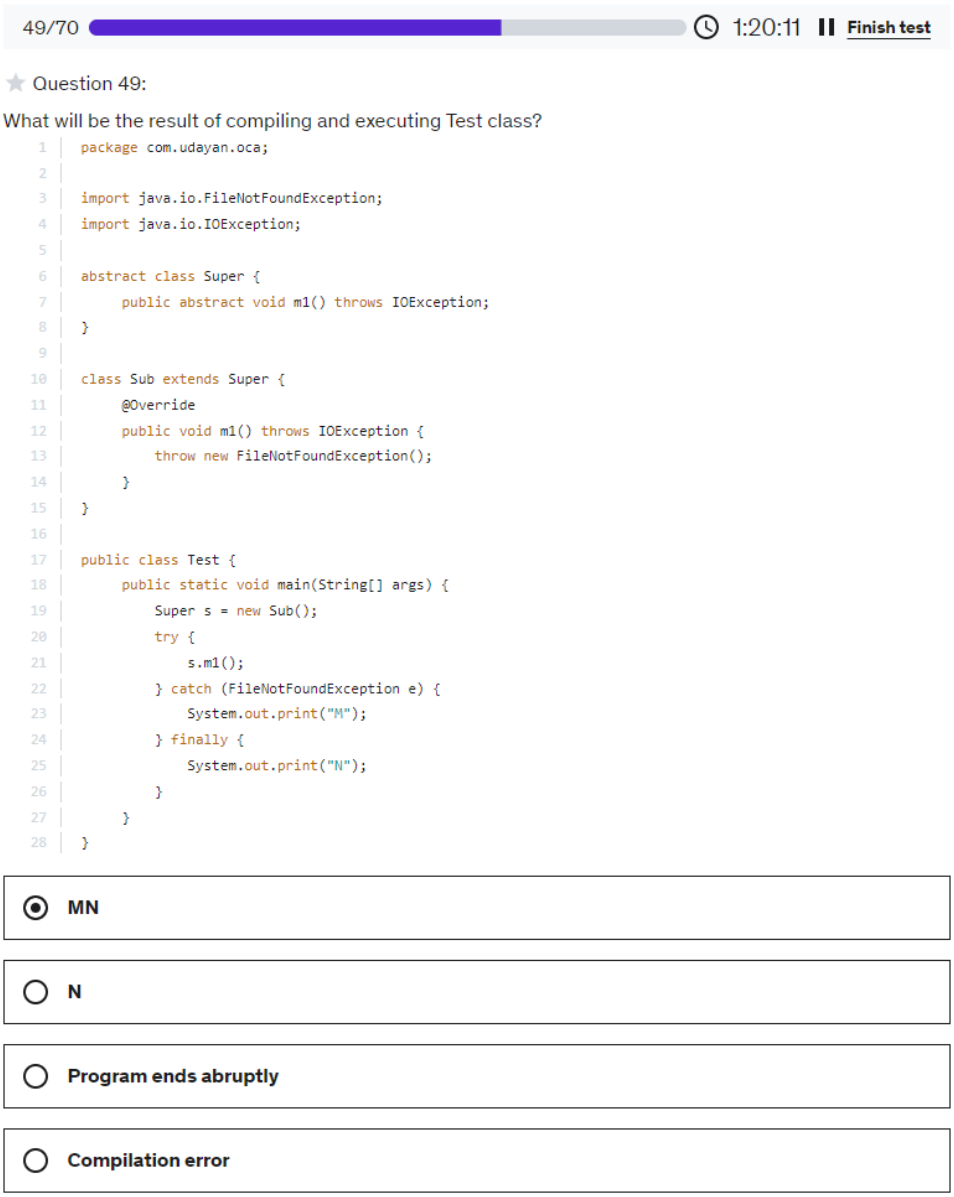

tor Build Run Tools VCS Window Help CodeWars - classC.java - Administrator
Test main
B.java C.java classA.java classB.java classC.java
class Test {
public static void main(String[] args) {
byte var = 100;
switch(var) {
case 100:
System.out.println("var is 100");
break;
case 200:
Sys
bre
default
Sys
}
}
}
Required type: byte
Provided: int
Cast to 'byte' Alt+Shift+Enter More actions... Alt+Enter

Super class throws IOException – Derived class can throw IOException or its child class Exceptio or not throw any exception.

But Derived class throw FileNotFoundException, either to be handled by try catch should declare throws – FileNotFoundException / IOException/ Exception or Throwable – can be used. But since Super class throws IOException- sub class cannot be clared using throws Exception or Throwable.

Hence we are left with IOException or FileNotFoundException. The below throws IOException which is the parent of FileNotFound which is ok – No issue with Derived class

Main Class – Has try catch for FileNotFoundException – but its not Handling IOException – So compile time error



-100

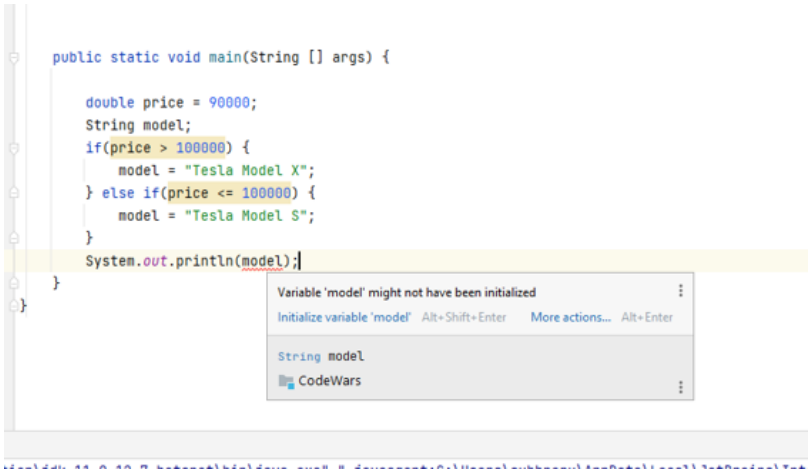
101

```
class Test {
    public static void main(String[] args) {


            int a = 100;
            System.out.println(-a++);
        System.out.println(a);

    }
}
```

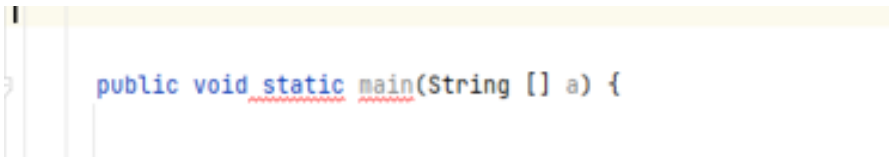
```
class Test1 {

    1 usage
    public void m1(int i) {
        System.out.println("int Version:"+i);

    }
    1 usage
    public void m1(char i) {
        System.out.println("char version:"+i);

    }

    public static void main(String args[]){

        Test1 t = new Test1();
           int i='5';
           t.m1(i);
           t.m1( i: '5');


       }}
```

```
Run:  Test1
"C:\Program Files\Eclipse Foundation\jdk-11.0.12.7-hotspot\b
int Version:53
char version:5
```

```java
public static void main(String args[]){

    Test1 t = new Test1();
       int i=5;
       double d = 5.0;
       if(i==d){
           System.out.println("Primitive type casting during expression evaluation: true"); // True
       }
       else{
           System.out.println("false");
       }

       Integer i1=5;
       Double d1 =5.0;

       if(i1==d1){// Compilation issue. Cant autounbox and typecasting at same time
        System.out.println("true");
       }
       else{
        System.out.println("Wrapper type casting during evaluation: false");
       }
}}
```



Void followed by method name



57/70 1:00:23 Finish test

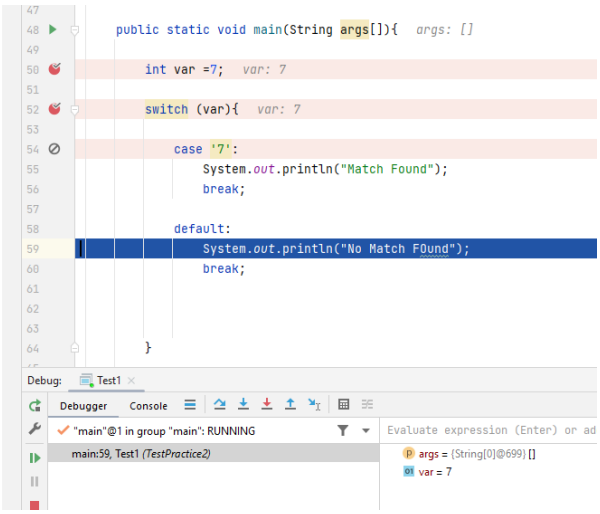
Question 57:

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.time.LocalDateTime;
5
6  public class Test {
7      public static void main(String [] args) {
8          LocalDateTime obj = LocalDateTime.now();
9          System.out.println(obj.getSecond());
10     }
11 }
```

Which of the following statement is correct?

- (●) **It will print any int value between 0 and 59**
- ( ) **It will print any int value between 1 and 60**
- ( ) **Code compiles successfully but throws Runtime exception**
- ( ) **Code fails to compile**



Var is int. Case Label ‘7’ char does not throw any error.

Executes Default

```
int var =7;

switch (var){

    case '7':
        System.out.println("Match Found");
        break;

    case '55':
        Sys                                    );
        bre
                Too many characters in character literal

    default:
        System.out.println("No Match FOund");
        break;
```

Question 1: **Incorrect**

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

```
package com.udayan.oca;

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");
        }
    }
}
```

- [x] `Character var = '7';` (Incorrect)
- [x] `Character var = 7;` (Correct)
- [x] `char var = 7;` (Correct)
- [x] `Integer var = 7;` (Correct)
- [x] `char var = '7';` (Incorrect)

Explanation

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.

Character var='7'
Here var-> Character. So Case 7: int 7 is converted to char '\u0007' Mismatches

Character var =7 -> int to Char -> '\u0007'
Case 7: int->Char -> '\u0007'

Char var =7; int to Char -> '\u0007'
Case 7: int->Char -> '\u0007'

Integer var=7;-> Int 7
Case 7: int 7

```
public static void main(String args[]){   args: []

    int i=55;   i: 55
    char c='7';   c: '7' 55
    if (i==c) {   i: 55    c: '7' 55

        System.out.println(Boolean.toString( b: true));

    }
```

```
    int var =55;

    switch (var){

        case '7':
            System.out.println("Match Found");
            break;
        default:
            System.out.println("No Match FOund");
            break;
    }
}}
```

Here char '7' in case label is converted to int 55 and comparison works

Character or char can always be converted to int . Even if the case label is char can be converted to int. Character comparison is always perofmed using int

Below char '7' is converted to int 55 and there happens to be duplicate value.. SO CE

```
    int var =55;

    switch (var){

        case '7':
            System.out.println("Match Found");
            break;

        case 9:
            System.out.println("Match Found");
            break;

        case 55:
            System.out.println("Match Found");
            break;
        default:
            System.out.println("No Match FOund");
            break;
    }
}}
```

```
int var =55;

char var1 =7;
Character var2 =7;

switch (var){

    case '6':
        System.out.println("Match Found");
        break;

    case '\u0007':
        System.out.println("Match Found");
        break;
    case 7:
        System.out.println("Match Found");
        break;

    case 55:
        System.out.println("Match Found");
        break;
    default:
        System.out.println("No Match FOund");
        break;
}
}}
```

‘7’ == 7

9 ==’9’

Will always fail

‘7’==55 -> int value of ‘7’ will pass.

Chars are converted to int for expression evaluation

List cant accept primitives. So primitive ints are converted to Integers. When

1. `E remove(int index)`: This method removes the element at the specified index and returns it. The subsequent elements are shifted to the left by one place. This method throws `IndexOutOfBoundsException` if the specified index is out of range. If the list implementations does not support this operation, `UnsupportedOperationException` is thrown.

2. `boolean remove(Object o)` This method removes the first occurrence of the specified `Object`. If the list doesn’t contain the given element, it remains unchanged. This method returns `true` if an element is removed from the list, otherwise `false`. If the object is `null` and list doesn’t support `null` elements, NullPointerException is thrown. `UnsupportedOperationException` is thrown if the list implementation doesn’t support this method.

[A,C,C,B,A]

[B]

```
List<String> list = new ArrayList<>();

list.add("A");
list.add("B");
list.add("C");
list.add("C");
list.add("B");
list.add("A");

System.out.println(list);

String removedStr = list.remove(1);
System.out.println(list);
System.out.println(removedStr);
```

Copy

```
List<String> list = new ArrayList<>();

list.add("A");

String removedStr = list.remove(10);
```

Copy

This code constructs a list with a length of `1`. However, when the code attempts to remove the element at index `10`:

```
Output
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index 10 out of bounds for lengt
        at java.base/jdk.internal.util.Preconditions.outOfBounds(Preconditions.java:64)
        at java.base/jdk.internal.util.Preconditions.outOfBoundsCheckIndex(Preconditions.java:70
        at java.base/jdk.internal.util.Preconditions.checkIndex(Preconditions.java:248)
        at java.base/java.util.Objects.checkIndex(Objects.java:372)
        at java.base/java.util.ArrayList.remove(ArrayList.java:535)
        at com.journaldev.java.ArrayListRemove.main(ArrayListRemove.java:19)
```

The `List.of()` method creates an immutable list, which can't be modified.

```
List<String> list = List.of("a", "b");

System.out.println(list);

String removedStr = list.remove(1);

System.out.println(removedStr);
```

Copy

Then the code attempts to use the `remove()` method to remove the element at index `1`:

```
$ Exception in thread "main" java.lang.UnsupportedOperationException
$       at java.base/java.util.ImmutableCollections.uoe(ImmutableCollections.java:142)
$       at java.base/java.util.ImmutableCollections$AbstractImmutableList.remove(ImmutableCollec
$       at TestRemoveList.main(TestRemoveList.java:12)
```

Copy

This attempt throws `UnsupportedOperationException`. It will also throw `UnsupportedOperationException`

IndexOutOfBound Excpetion

```
public static void main(String args[]){



    List<Integer> list = new ArrayList<>();

    list.add(100);
    list.add(200);
    list.add(300);
    System.out.println(list);

    list.remove( index: 100);

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

E remove(int index) -> method is overridden to remove the element at index.

Causes index out of bound exception

Since remove(int)- is a valid method it does not perform autoboxing in this case

Question 22: **Incorrect**

Given code:

```
1  package com.udayan.oca;
2
3  public class Test {
4      public static void main(String[] args) {
5          String [] arr = {"I", "N", "S", "E", "R", "T"};
6          for(/*INSERT*/) {
7              if (n % 2 == 0) {
8                  continue;
9              }
10             System.out.print(arr[n]); //Line n1
11         }
12     }
13 }
```

And below options:

1. int n = 0; n < arr.length; n += 1
2. int n = 0; n <= arr.length; n += 1
3. int n = 1; n < arr.length; n += 2
4. int n = 1; n <= arr.length; n += 2

How many above options can be used to replace /*INSERT*/, such that on execution, code will print NET on to the console?

N E T – index 1 , 3 , 5 % 2 != 0 will be printed

0,2,4 nos % 2==0 so these nos will not be printed

1. 0,1,2,3,4,5 -> N E T will be printed
2. 0,1,2,3,4,5,6 -> Here arr[6] is not called . so no run time error.
3. 1,3,5
4. 1,3,5

All 4 are valid methods

Question 32: **Incorrect**

Consider codes below:

```
1  //A.java
2  package com.udayan.oca;
3
4  public class A {
5       public int i1 = 1;
6       protected int i2 = 2;
7  }
```

```
1  //B.java
2  package com.udayan.oca.test;
3
4  import com.udayan.oca.A;
5
6  public class B extends A {
7       public void print() {
8           A obj = new A();
9           System.out.println(obj.i1); //Line 8
10          System.out.println(obj.i2); //Line 9
11          System.out.println(this.i2); //Line 10
12          System.out.println(super.i2); //Line 11
13      }
14
15      public static void main(String [] args) {
16          new B().print();
17      }
18 }
```

One of the statements inside print() method is causing compilation failure. Which of the below solutions will help to resolve compilation error?

- ○ **Comment the statement at Line 10**
- ◉ **Comment the statement at Line 8** (Incorrect)
- ○ **Comment the statement at Line 9** (Correct)
- ○ **Comment the statement at Line 11**

Explanation

Current object is
Of type B which is of child class ref
And can access protected member of parent class

Obj.i1 is accessing public member-valid

Obj.i2 – Parent Ref accessing protected variable- Throws error in Derived class of diff package

This.i2-> current object which is child ref and is trying to i2 which is super class pro variable – valid

super.i2 -> current object which is child ref of A can access super class i2 - Valid

List.remove(int) – will always look for index value of int no matter what objects the list holds

List.remove(char) – will be converted to int (ASCII value of the character) and performs index retrieval

Question 35: **Incorrect**

Consider code of Test.java file:

```
1  package com.udayan.oca;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  public class Test {
7      public static void main(String[] args) {
8          List<Character> list = new ArrayList<>();
9          list.add(0, 'V');
10         list.add('T');
11         list.add(1, 'E');
12         list.add(3, 'O');
13
14         if(list.contains('O')) {
15             list.remove('O');
16         }
17
18         for(char ch : list) {
19             System.out.print(ch);
20         }
21     }
22 }
```

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

- (●) **VET** (Incorrect)
- ( ) **VTEO**
- ( ) **Compilation error**
- ( ) **VTE**
- ( ) **VETO**
- ( ) **Runtime exception** (Correct)

list.contains('O') => char 'O' is auto-boxed to Character object and as Character class overrides equals(String) method this expression returns true. Control goes inside if-block and executes: list.remove('O');.

remove method is overloaded: remove(int) and remove(Object). char can be easily assigned to int so compiler tags remove(int) method. list.remove(<ASCCI value of 'O'>); ASCCI value of 'A' is 65 (this everybody knows) so ASCII value of 'O' will be more than 65.

list.remove('O') throws runtime exception, as it tries to remove an item from the index greater than 65 but allowed index is 0 to 3 only.

Question 38: **Incorrect**

Below is the code of Test.java file:

```
1  package com.udayan.oca;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  public class Test {
7       public static void main(String [] args) {
8           List<Integer> list = new ArrayList<Integer>();
9           list.add(new Integer(2));
10          list.add(new Integer(1));
11          list.add(new Integer(0));
12
13          list.remove(list.indexOf(0));
14
15          System.out.println(list);
16      }
17 }
```

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

- ○ **An exception is thrown at runtime**
- ○ **Compilation error**
- ○ **[2, 1]** **(Correct)**
- ◉ **[1, 0]** **(Incorrect)**

Explanation

remove method of List interface is overloaded: remove(int) and remove(Object).

indexOf method accepts argument of Object type, in this case list.indexOf(0) => 0 is auto-boxed to Integer object so no issues with indexOf code. list.indexOf(0) returns 2 (index at which 0 is stored in the list). So list.remove(list.indexOf(0)); is converted to list.remove(2);

Question 42: **Incorrect**

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

```
1  package com.udayan.oca;
2
3  public class Test {
4      public static void main(String[] args) {
5          Double [] arr = new Double[2];
6          System.out.println(arr[0] + arr[1]);
7      }
8  }
```

- ○ **ClassCastException is thrown at runtime**
- ◉ **0.0** **(Incorrect)**
- ○ **NullPointerException is thrown at runtime** **(Correct)**
- ○ **Compilation error**

Arr[0]+”” will print null as toString() method will be call

Arr[0]+Arr[1] is an expression and tries to return the XXXValue() and throw RunTime Exception

To String is perfomed in first three cases and hence null

```
Character[] list = new Character[2];
System.out.println(list[0]); // null

System.out.println(list[1]); // null
System.out.println(list[0]+""); // null
System.out.println(list[0]+'c'); // here trying to perform evaluation. Expression.
//Character.charValue() Method has to be called. null.charValue -> Throws null pointer Exception
```

When an expression has to evaluated. Wrapper.xxxValue() will be called and on Null ref will throw null pointer exception

```
public static void main(String args[]){


    Double[] list = new Double[2];
    System.out.println(list[0]); // null

    System.out.println(list[1]); // null
    System.out.println(list[0]+""); // null
    System.out.println(list[0]+'c'); // here trying to perform evaluation. Expression.
    //Character.charValue() Method has to be called. null.charValue -> Throws null pointer Exception
```

```
System.out.println(Integer.valueOf( s: "null"));//accepts String but cant be converted to Integer
// this is equal to Integer.parstInt(s); this cant be converted to Integer - null pointer Exception

System.out.println(Character.valueOf( c: null));// CE - Expects Char not null
System.out.println(Character.valueOf( c: "null"));//CE- Expects char not String

System.out.println(Boolean.valueOf( s: null));// valid accepts String if and evaluates to false
System.out.println(Boolean.valueOf( s: "null")); // false
```

```
System.out.println(Boolean.valueOf( s: null)+1);
```

```
C:\Users\subhperu\IdeaProjects\CodeWars\src\TestPractice2\classC.java:51:
java: bad operand types for binary operator '+'
  first type:  java.lang.Boolean
  second type: int
```

Null pointer Excpetion for below

```
public static void main(String args[]){

    Integer i =null;
     System.out.println(i+1);
```

```
3   class Student {
4       String name;
5       int age;
6
7       void Student() {
8           Student("James", 25);
9       }
10
11      void Student(String name, int age) {
12          this.name = name;
13          this.age = age;
14      }
15  }
16
17  public class Test {
18      public static void main(String[] args) {
19          Student s = new Student();
20          System.out.println(s.name + ":" + s.age);
21      }
22  }
```

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

- ◯ **null:0** (Correct)
- ◯ **An exception is thrown at runtime**
- ◯ **James:25**
- ◉ **Compilation error** (Incorrect)

- [x] `e -> e.getSalary() >= 10000` **(Correct)**
- [ ] `(e) -> { e.getSalary() >= 10000; }`
- [ ] `(Employee e) -> { return e.getSalary() >= 10000; }` **(Correct)**
- [ ] `e -> { e.getSalary() >= 10000 }`
- [x] `e - > e.getSalary() >= 10000` **(Incorrect)**

Explanation
Jack's salary is 5000 and Liya's salary is 8000. If Employee's salary is >= 10000 then that Employee object is removed from the list.

Allowed lambda expression is:

(Employee e) -> { return e.getSalary() >= 10000; },

Can be simplified to: (e) -> { return e.getSalary() >= 10000; } => type can be removed from left side of the expression.

Further simplified to: e -> { return e.getSalary() >= 10000; } => if there is only one parameter in left part, then round brackets (parenthesis) can be removed.

Further simplified to: e -> e.getSalary() >= 10000 => if there is only one statement in the right side then semicolon inside the body, curly brackets and return statement can be removed. But all 3 [return, {}, ;] must be removed together.

NOTE: there should not be any space between - and > of arrow operator.

Question 20: Skipped

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.time.LocalDate;
5  import java.time.LocalTime;
6
7  public class Test {
8       public static void main(String [] args) {
9           LocalDate date = LocalDate.parse("1947-08-14");
10          LocalTime time = LocalTime.MAX;
11          System.out.println(date.atTime(time));
12      }
13 }
```

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

- ◯ **1947-08-14T23:59:59.999999999** **(Correct)**
- ◯ **1947-08-14T23:59:59.999**
- ◯ **1947-08-14T23:59:59.0**
- ◯ **1947-08-14T23:59:59**

Explanation

LocalTime.MIN --> {00:00}, LocalTime.MAX --> {23:59:59.999999999}, LocalTime.MIDNIGHT --> {00:00}, LocalTime.NOON --> {12:00}.

date.atTime(LocalTime) method creates a LocalDateTime instance by combining date and time parts.

toString() method of LocalDateTime class prints the date and time parts separated by T in upper case.

Question 34: Skipped

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.time.LocalDate;
5
6  public class Test {
7      public static void main(String [] args) {
8          LocalDate newYear = LocalDate.of(2018, 1, 1);
9          LocalDate christmas = LocalDate.of(2018, 12, 25);
10         boolean flag1 = newYear.isAfter(christmas);
11         boolean flag2 = newYear.isBefore(christmas);
12         System.out.println(flag1 + ":" + flag2);
13     }
14 }
```

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

- ◯ **false:true** (Correct)
- ◯ **true:false**
- ◯ **Compilation error**
- ◯ **An exception is thrown at runtime**

Explanation

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.

Question 70: Skipped

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  public class Test {
5      public static void main(String[] args) {
6          StringBuilder sb = new StringBuilder(100);
7          System.out.println(sb.length() + ":" + sb.toString().length());
8      }
9  }
```

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

- ◯ **100:100**
- ◯ **16:16**
- ◯ **100:0**
- ◯ **16:0**
- ◯ **0:0** **(Correct)**

Explanation

`new StringBuilder(100);` creates a StringBuilder instance, whose internal char array's length is 100 but length() method of StringBuilder object returns the number of characters stored in the internal array and in this case it is 0. So, `sb.length()` returns 0.

sb.toString() is the String representation of StringBuilder instance and in this case as there are no characters inside the StringBuilder object, hence sb.toString() returns an empty String "", so `sb.toString().length()` also returns 0.

Output is 0:0.

Press Esc to exit full screen

Question 5: **Correct**

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.util.ArrayList;
5  import java.util.List;
6
7  public class Test {
8       public static void main(String[] args) {
9           List<StringBuilder> dryFruits = new ArrayList<>();
10          dryFruits.add(new StringBuilder("Walnut"));
11          dryFruits.add(new StringBuilder("Apricot"));
12          dryFruits.add(new StringBuilder("Almond"));
13          dryFruits.add(new StringBuilder("Date"));
14
15          for(int i = 0; i < dryFruits.size(); i++)
16          {
17              if(i == 0) {
18                  dryFruits.remove(new StringBuilder("Almond"));
19              }
20           }
21
22          System.out.println(dryFruits);
23      }
24 }
```

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

- ◯ **[Walnut, Apricot, Date]**
- ◯ **An exception is thrown at runtime**
- ◯ **[Walnut, Date]**
- ◉ **[Walnut, Apricot, Almond, Date]** (Correct)

Explanation

In this example, code is trying to remove an item from the list while iterating using traditional for loop so one can think that this code would throw java.util.ConcurrentModificationException.

But note, java.util.ConcurrentModificationException will never be thrown for traditional for loop. It is thrown for for-each loop or while using Iterator/ListIterator.

In this case dryFruits.remove(new StringBuilder("Almond")); will never remove any items from the list as StringBuilder class doesn't override the equals(Object) method of Object class.

StringBuilder instances created at "dryFruits.add(new StringBuilder("Almond"));" and "dryFruits.remove(new StringBuilder("Almond"));" are at different memory locations and equals(Object) method returns false for these instances.

Question 6: **Correct**

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

```
1  package com.udayan.oca;
2
3  public class Test {
4       public static void main(String[] args) {
5           for:
6           for (int i = 2; i <= 100; i = i + 2) {
7               for(int j = 1; j <= 10; j++) {
8                   System.out.print(i * j + "\t");
9               }
10              System.out.println();
11              if(i == 10) {
12                  break for;
13              }
14          }
15      }
16 }
```

- ◯ **Total 100 rows will be there in the output**
- ◯ **Total 50 rows will be there in the output**
- ◉ **Compilation error** (Correct)
- ◯ **Total 5 rows will be there in the output**

Explanation

for is a keyword and hence can't be used as a label.

Java labels follow the identifier naming rules and one rule is that we can't use java keywords as identifier. Hence, Compilation error.

Interface methods are by default public and abstract

Question 7: **Incorrect**

For the given code:

```
1  package com.udayan.oca;
2
3  interface I01 {
4      void m1();
5  }
6
7  public class Implementer extends Object implements I01{
8      protected void m1() {
9
10     }
11 }
```

- (●) **interface I01 gives compilation error as method m1 is not public.** (Incorrect)
- ( ) **None of the other options.**
- ( ) **Implementer class declaration is not correct.**
- ( ) **Method m1() in Implementer class is not implemented correctly.** (Correct)

Explanation

void m1(); in interface I01 is equivalent to public abstract void m1(); So method m1() is implicitly public and abstract.

In java, a class can extend from only one class but can implement multiple interfaces. Correct keywords are: extends and implements.

so class declaration is correct.

As method m1() is implicitly public in I01, hence overriding method in Implementer class should also be public. But it is protected and hence compiler complains.

Question 8: Skipped

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

```
1  //TestBaseDerived.java
2  package com.udayan.oca;
3
4  class Base {
5      protected void m1() {
6          System.out.println("Base: m1()");
7      }
8  }
9
10 class Derived extends Base {
11     void m1() {
12         System.out.println("Derived: m1()");
13     }
14 }
15
16 public class TestBaseDerived {
17     public static void main(String[] args) {
18         Base b = new Derived();
19         b.m1();
20     }
21 }
```

- ◯ **Base: m1()**
- ◯ **None of the other options** (Correct)
- ◯ **Derived: m1()**
- ◯ **Base: m1()**
  Derived: m1()

Explanation

Derived class overrides method m1() of Base class. Access modifier of method m1() in Base class is protected, so overriding method can use protected or public.

But overriding method in this case used default modifier and hence there is compilation

The **java instanceof operator** is used to test whether the object is an instance of the specified type (class or subclass or interface).

The instanceof in java is also known as type *comparison operator* because it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that has null value, it returns false.

---

instanceof in java with a variable that have null value

If we apply instanceof operator with a variable that have null value, it returns false. Let's see the example given below where we apply instanceof operator with the variable that have null value.

```
class Dog2{
 public static void main(String args[]){
  Dog2 d=null;
  System.out.println(d instanceof Dog2);//false
 }
}
```


```
class M { }
class N extends M { }
class O extends N { }
class P extends O { }

public class Test {
     public static void main(String args []) {
         M obj = new O();
         if(obj instanceof M)
           System.out.print("M");
         if(obj instanceof N)
           System.out.print("N");
         if(obj instanceof O)
           System.out.print("O");
         if(obj instanceof P)
           System.out.print("P");
     }
}
```

- ◯ **MNO** (Correct)
- ◯ **MOP**
- ◯ **NOP**
- ◯ **MNP**

Explanation

M

^

N

^

O [obj refers to instance of O class]

^

P

obj instanceof M -> true

obj instanceof N -> true

obj instanceof O -> true

but

obj instanceof P -> false

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

```
1  package com.udayan.oca;
2
3  public class Test {
4       public static void main(String[] args) {
5           m(1);
6       }
7
8       private static void m(Object obj) {
9           System.out.println("Object version");
10      }
11
12      private static void m(Number obj) {
13          System.out.println("Number version");
14      }
15
16      private static void m(Double obj) {
17          System.out.println("Double version");
18      }
19 }
```

- ◯ **Compilation error**
- ◯ **Double version**
- ◯ **Number version** (Correct)
- ◯ **Object version**

Explanation

There are 3 overloaded method m. Note all the numeric wrapper classes (Byte, Short, Integer, Long, Float and Double) extend from Number and Number extends from Object.

Compiler either does implicit casting or Wrapping but not both. 1 is int literal, Java compiler can't implicit cast it to double and then box it to Double rather it boxes i to Integer and as Number is the immediate super class of Integer so Number version refers to Integer object.

Number version is printed on to the console.

StringBuffer.delete(5,6) substring of (5-6) that is only 5

Question 3: **Incorrect**

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

```
1  package com.udayan.oca;
2
3  public class Test {
4      public static void main(String[] args) {
5          StringBuilder sb = new StringBuilder("SpaceStation");
6          sb.delete(5, 6).insert(5, " S").toString().toUpperCase();
7          System.out.println(sb);
8      }
9  }
```

- (●) **SPACE STATION** (Incorrect)
- ( ) **Space Station** (Correct)
- ( ) **SPACE SATION**
- ( ) **Space Sation**

Explanation

sb - > "SpaceStation"

sb.delete(5, 6) -> "Spacetation"

sb.insert(5, " S") -> "Space Station"

sb.toString() -> Creates a new String object "Space Station"

"Space Station".toUpperCase() -> Creates another String object "SPACE STATION" but the String object is not referred and used.

Method invocation on sb modifies the same object, so after insert(5, " S") method invocation sb refers to "Space Station" and this is printed to the Console.

Question 4: Skipped

Consider below code:

```
1  package com.udayan.oca;
2
3  public class Test {
4       static Double d1;
5       static int x = d1.intValue();
6
7       public static void main(String[] args) {
8           System.out.println("HELLO");
9       }
10 }
```

On execution, does Test class print "HELLO" on to the console?

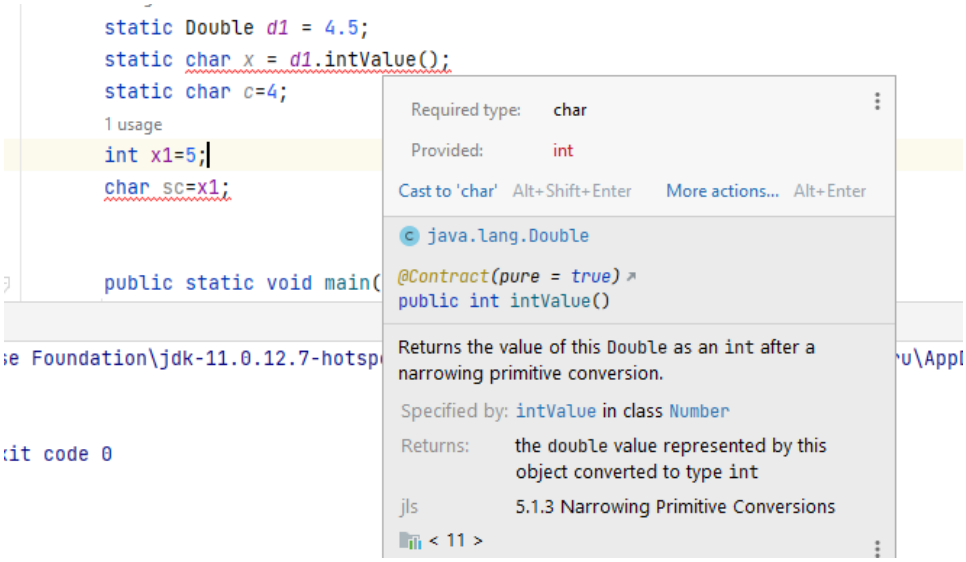
- ◯ **Yes, HELLO is printed on to the console**
- ◯ **No, HELLO is not printed on to the console** (Correct)

Explanation

To invoke the special main method, JVM loads the class in the memory. At that time, static fields of Test class are initialized. d1 is of Double type so null is assigned to it.

x is also static variable so d1.intValue(); is executed and as d1 is null hence d1.intValue() throws a NullPointerException and as a result an instance of java.lang.ExceptionInInitializerError is thrown.

```
class Test1 {

    1 usage
    static Double d1;
    static int d2 = d1.intValue();
    public static void main(String args[]){
        System.out.println();// false
    }}
```



Converting a primitive value (an `int`, for example) into an object of the corresponding wrapper class (`Integer`) is called autoboxing. The Java compiler applies autoboxing when a primitive value is:

- Passed as a parameter to a method that expects an object of the corresponding wrapper class.
- Assigned to a variable of the corresponding wrapper class.

Converting an object of a wrapper type (`Integer`) to its corresponding primitive (`int`) value is called unboxing. The Java compiler applies unboxing when an object of a wrapper class is:

- Passed as a parameter to a method that expects a value of the corresponding primitive type.
- Assigned to a variable of the corresponding primitive type.

Char to int widening, Int to char is narrowing should be explicitly casting is req

1. **double** -> **float** -> **long** -> **int** -> **char** -> **short** -> **byte**

Compile by: javac IntToCharExample1.java

CharExample1.java:4: error: incompatible types: possible lossy conversion from int to char
char c=a;
^
1 error

```
3
4   import java.time.LocalDate;
5   import java.time.Period;
6
7   public class Test {
8       public static void main(String [] args) {
9           LocalDate obj = new LocalDate(2020, 2, 14);
10          System.out.println(obj.minus(Period.ofDays(10)));
11      }
12  }
```

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

- ○ **2020-02-04**
- ○ **Runtime exception**
- ○ **2020-02-03**
- ○ **Compilation error** **(Correct)**

Explanation
Constructor of LocalDate is declared private so cannot be called from outside, hence new LocalDate(2020, 2, 14); causes compilation failure.

Overloaded static methods "of" and "parse" are provided to create the instance of LocalDate.

LocalTime, LocalDateTime, Period also specify private constructors and provide "of" and "parse" methods to create respective instances.

Question 9: Skipped

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.time.LocalDate;
5  import java.time.Month;
6  import java.time.Period;
7
8  public class Test {
9      public static void main(String [] args) {
10         LocalDate date = LocalDate.of(2000, Month.JANUARY, 1);
11         Period period = Period.parse("p-30000y");
12         System.out.println(date.plus(period));
13     }
14 }
```

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

- ◯ **28000-01-01**
- ◯ **32000-01-01**
- ◯ **Runtime exception**
- ◯ **Compilation error**
- ◯ **-28000-01-01** **(Correct)**

Explanation

There are 2 of methods available in LocalDate class: of(int, int, int) and of(int, Month, int). Month can either be passed as int value (1 to 12) or enum constants Month.JANUARY to Month.DECEMBER.

Period.parse(CharSequence) method accepts the String parameter in "PnYnMnD" format, over here P,Y,M and D can be in any case. "p-30000y" means Period of -30000 years.

The minimum supported LocalDate is: {-999999999-01-01} and maximum supported LocalDate is: {+999999999-12-31}. If period of -30000 years is added to 1st Jan 2000, then result is 1st Jan -28000.

```java
public class Test {
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<>();
        list.add(100);
        list.add(7);
        list.add(50);
        list.add(17);
        list.add(10);
        list.add(5);

        list.removeIf(a -> a % 10 == 0);

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

- ◯ **[100, 7, 50, 17, 10, 5]**
- ◯ **Compilation error**
- ◯ **Runtime Exception**
- ◯ **[7, 17, 5]** **(Correct)**
- ◯ **[100, 50, 10]**

Explanation

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.

Predicate's test method returns true for all the Integers divisible by 10.

Loop 1-10

Will be skipped if start%2 ==0

So 2,4,6,8,10 will be skipped. 1,3,5,7,9 -> 25

Question 12: **Skipped**

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

```
1  package com.udayan.oca;
2
3  public class Test {
4       public static void main(String[] args) {
5           int start = 1;
6           int sum = 0;
7           do {
8               if(start % 2 == 0) {
9                   continue;
10              }
11              sum += start;
12          } while(++start <= 10);
13          System.out.println(sum);
14      }
15 }
```

- ○ **24**
- ○ **25** **(Correct)**
- ○ **55**
- ○ **Compilation error**

LocalDate. LocaDateTime .LocalTime or immutable . Here the date value is not changed

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.time.LocalDate;
5
6  public class Test {
7       public static void main(String [] args) {
8           LocalDate date = LocalDate.parse("2000-06-25");
9           while(date.getDayOfMonth() >= 20) {
10              System.out.println(date);
11              date.plusDays(-1);
12          }
13      }
14 }
```

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

- ◯ **System.out.println(date); is executed 6 times**
- ◯ **An exception is thrown at runtime**
- ◯ **Compilation error**
- ◯ **System.out.println(date); is executed more than 6 times** (Correct)

Explanation

date --> {2000-06-25}. date.getDayOfMonth() = 25, 25 >= 20 is true, hence control goes inside while loop and executes System.out.println(date); statement.

date.plusDays(-1); creates a new LocalDate object {2000-06-24} but date reference variable still refers to {2000-06-25}. date.getDayOfMonth() again returns 25, this is an infinite loop.

```java

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

class Student {
     private String name;
     private int age;

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

     public String toString() {
         return "Student[" + name + ", " + age + "]";
     }

     public boolean equals(Student obj) {
         if(obj instanceof Student) {
             Student stud = (Student)obj;
             if(this.name.equals(stud.name) && this.age == stud.age) {
                 return true;
             }
         }
         return false;
     }
}

public class Test {
     public static void main(String[] args) {
         List<Student> students = new ArrayList<>();
         students.add(new Student("James", 25));
         students.add(new Student("James", 27));
         students.add(new Student("James", 25));
         students.add(new Student("James", 25));

         students.remove(new Student("James", 25));

         for(Student stud : students) {
             System.out.println(stud);
         }
     }
}
```

- Student[James, 27]
- Student[James, 25]
Student[James, 27]
Student[James, 25]
Student[James, 25] (Correct)
- Student[James, 27]
Student[James, 25]
Student[James, 25]

Explanation

Before you answer this, you must know that there are 5 different Student object created in the memory (4 at the time of adding to the list and 1 at the time of removing from the list). This means these 5 Student objects will be stored at different memory addresses.

remove(Object) method removes the first occurrence of matching object and equals(Object) method decides whether 2 objects are equal or not. equals(Object) method has NOT been overridden by the Student class. In fact, equals(Student) is overloaded. But overloaded version is not invoked while equating the Student objects.

equals(Object) method defined in Object class is invoked and equals(Object) method defined in Object class uses == operator to check the equality and in this case as all the Student objects are stored at different memory location, hence not equal.

Nothing is removed from the students list, all the 4 Student objects are printed in the insertion order.

remove

`public boolean remove(Object o)`

Removes the first occurrence of the specified element from this list, if it is present. If the list does not contain the element, it is unchanged. More formally, removes the element with the lowest index `i` such that `(o==null ? get(i)==null : o.equals(get(i)))` (if such an element exists). Returns `true` if this list contained the specified element (or equivalently, if this list changed as a result of the call).

Specified by:

`remove` in interface `Collection<E>`

Specified by:

`remove` in interface `List<E>`

Overrides:

`remove` in class `AbstractCollection<E>`

Parameters:

`o` - element to be removed from this list, if present

Returns:

`true` if this list contained the specified element

Question 15: Skipped

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

```
1  package com.udayan.oca;
2
3  public class Bonus {
4       public static void main(String[] args) {
5           int $ = 80000;
6           String msg = ($ >= 50000) ? "Good bonus" : "Average bonus";
7           System.out.println(msg);
8       }
9  }
```

- ◯ **Compilation error**
- ◯ **Good bonus** (Correct)
- ◯ **Average bonus**

Explanation

$ is valid identifier. $ = 80000

This is an example of ternary operator. First operand ($ >= 50000) is a boolean expression which is true, as 80000 >= 50000 is true.

msg will refer to "Good bonus".

Question 18: Skipped

Which of the following correctly imports Animal class from com.masaimara package?

- ○ `import com.masaimara;`
- ○ `Import com.masaimara.Animal;`
- ○ `import com.masaimara.*;` (Correct)
- ○ `Import com.masaimara.*;`

Explanation

Following import statements are correct:

import com.masaimara.*;

import com.masaimara.Animal;

NOTE: all small case letters in import keyword.

Question 20: Skipped

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

```
1  package com.udayan.oca;
2
3  public class Test {
4      public static void main(String[] args) {
5          String [] arr = new String[1];
6          System.out.println(arr[0].isEmpty());
7      }
8  }
```

- ○ **true**
- ○ **ArrayIndexOutOfBoundsException is thrown at runtime**
- ○ **false**
- ○ **NullPointerException is thrown at runtime** (Correct)

Explanation

All the elements of array are initialized to respective zeros (in case of primitive type) or null (in case of reference type).

So, arr[0] refers to null.

Method 'isEmpty()' is invoked on null reference and hence NullPointerException is thrown at runtime.

Again LocalDate LocalTime and LocalDateTime are immutable

Question 23: **Skipped**

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.time.LocalDate;
5
6  public class Test {
7       public static void main(String [] args) {
8           LocalDate date = LocalDate.parse("2018-06-06");
9           date.minusDays(10);
10          System.out.println(date);
11      }
12 }
```

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

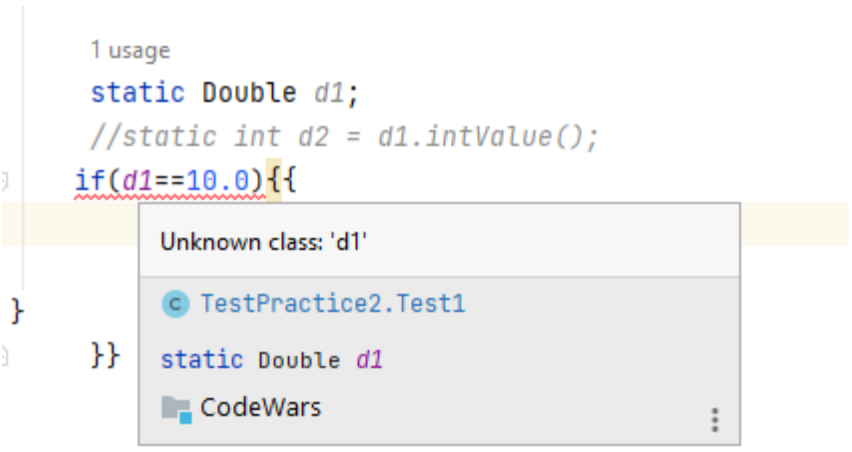
- ◯ **2018-06-06** **(Correct)**
- ◯ **2018-05-26**
- ◯ **2018-06-25**
- ◯ **2018-05-27**
- ◯ **2018-06-26**

Question 24: **Skipped**

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

```
1  package com.udayan.oca;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  public class Test {
7       public static void main(String[] args) {
8           List<Integer> list = new ArrayList<>();
9           list.add(110);
10          list.add(new Integer(110));
11          list.add(110);
12
13          list.removeIf(i -> i == 110);
14          System.out.println(list);
15      }
16 }
```

- ○ **[110, 110, 110]**
- ○ **[110]**
- ○ **[]** **(Correct)**
- ○ **[110, 110]**



Null Pointer Exception, when doing Double.IntValue();

```
class Test1 {

    1 usage
    static Double d1=null;
    //static int d2 = d1.intValue();

    public static void main(String ars[]){


        if(d1 ==10.8){
            System.out.println("Match Found");
        }
```

Consider 2 files:

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

public class Counter {
    public int count = 0;

    public Counter(int start) {
        count = start;
    }

    public int getCount() {
        return count;
    }

    public void increase(int val) {
        count = count + val;
    }

    public String toString() {
        return this.count + "";
    }
}
```

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

import java.util.Arrays;

import com.udayan.oca.Counter;

public class Test {
    public static void main(String[] args) {
        Counter[] arr = new Counter[] { new Counter(-1000), new Counter(539),
new Counter(0) };

        /* INSERT */

        System.out.println(Arrays.toString(arr));
    }
}
```

A number can be made 100 by subtracting the no with the same no and adding 100 to it

500+ (-500+100) or
500 + (100 – 500)
-2000
(-2000)+(100-(-2000))
-2000+2100=100

Explanation

There are 2 ways to change the value of count variable of Counter class:

1. As access modifier of count variable is public, hence it can easily be accessed from other classes using the instance of Counter class, such as `new Counter().count` or `obj.count` (where obj is reference variable of Counter type, referring to Counter variable's instance)

2. By invoking the increase(int) method of Counter class.

Now let's check all the blocks one by one:

1.

```
for(Counter ctr : arr) {
   ctr.count = 100;
}
```

✓ It will assign 100 to count variables of three instances of Counter class.

2.

```
for (Counter ctr : arr) {
   int x = ctr.getCount();
   x = 100;
}
```

✗ x is local variable and is copy of ctr.count. Hence, assigning 100 to x will not affect the value of ctr.count.

3.

```
for (Counter ctr : arr) {
   ctr.getCount() = 100;
}
```

✗ ctr.getCount() returns int value and not a variable, hence cannot be used on left side of assignment operator. It causes compilation error.

4.

```
for(Counter ctr : arr) {
   ctr.increase(100 - ctr.count);
}
```

✓ You must have noticed that value of count variable of 3 array elements are: -1000, 539, 0. How will you change all 3 values to 100 using same expression? It is by adding 100 and subtracting current value. For example,

-1000 + 100 -(-1000) = 100

or

539 + 100 - 539 = 100

or

0 + 100 - 0 = 100

And same this is done by executing `ctr.increase(100 - ctr.count);` statement.

5.

```
for (Counter ctr : arr) {
   ctr.increase(100 - ctr.getCount());
}
```

✓ Same as block no. 4. Only difference is ctr.getCount() is used instead of ctr.count.

6.

```
for(Counter ctr : arr) {
   ctr.increase(-ctr.getCount() + 100);
}
```

✓ Same as block no. 5.

7.

```
for(Counter ctr : arr) {
   ctr.increase(-ctr.count + 100);
}
```

```
13
14          System.out.println(Arrays.toString(arr));
15      }
16  }
```

Currently on executing Test class, output is: [-1000, 539, 0].

And below blocks:

1.

```
for(Counter ctr : arr) {
   ctr.count = 100;
}
```

2.

```
for (Counter ctr : arr) {
   int x = ctr.getCount();
   x = 100;
}
```

3.

```
for (Counter ctr : arr) {
   ctr.getCount() = 100;
}
```

4.

```
for(Counter ctr : arr) {
   ctr.increase(100 - ctr.count);
}
```

5.

```
for (Counter ctr : arr) {
   ctr.increase(100 - ctr.getCount());
}
```

6.

```
for(Counter ctr : arr) {
   ctr.increase(-ctr.getCount() + 100);
}
```

7.

```
for(Counter ctr : arr) {
   ctr.increase(-ctr.count + 100);
}
```

Runtime exceptions can occur anywhere in a program, and in a typical one they can be very numerous. Having to add runtime exceptions in every method declaration would reduce a program's clarity. Thus, **the compiler does not require that you catch or specify runtime exceptions** (although you can).

Question 27: **Skipped**

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

```
1  package com.udayan.oca;
2
3  public class Test {
4      private static void m1() {
5          System.out.println(1/0);
6      }
7
8      public static void main(String[] args) {
9          try {
10             m1();
11         } finally {
12             System.out.println("A");
13         }
14     }
15 }
```

- ◯ **A is printed to the console, stack trace is printed and then program ends abruptly.** (Correct)
- ◯ **Compilation error.**
- ◯ **A is printed to the console and program ends normally.**
- ◯ **A is printed to the console, stack trace is printed and then program ends normally.**

Explanation

Method m1() throws an instance of ArithmeticException and method m1() doesn't handle it, so it forwards the exception to calling method main.

Method main doesn't handle ArithmeticException so it forwards it to JVM, but just before that finally block is executed. This prints A on to the console.

After that JVM prints the stack trace and terminates the program abruptly.

Don’t over see , check all details return type and the value returned
Int -> null . null is not a valid int value

Question 28: **Skipped**

Consider code of Test.java file:

```
1  package com.udayan.oca;
2
3  public class Test {
4      public static void main(String [] args) {
5          int [] arr = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
6           System.out.println(process(arr, 3, 8)); //Line 5
7      }
8
9      /*INSERT*/
10 }
```

Line 5 is giving compilation error as process method is not found.

Which of the following method definitions, if used to replace /*INSERT*/, will resolve the compilation error?

Select 3 options.

☐
```
1  private static String process(int [] arr, int start, int end) {
2      return null;
3  }
```
(Correct)

☐
```
1  private static int[] process(int [] arr, int start, int end) {
2      return null;
3  }
```
(Correct)

☐
```
1  private static String[] process(int [] arr, int start, int end) {
2      return null;
3  }
```
(Correct)

☐
```
1  private static int process(int [] arr, int start, int end) {
2      return null;
3  }
```

Here double 10.5 cant be implicitly converted to int so, compiler converts int to double implicity
And the value would be 10.0 so not equals.

```
if(10.5 == 10){
    System.out.println("Match");
}
```

List.set -> Replcaes the element

Question 32: **Skipped**

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.util.ArrayList;
5  import java.util.List;
6
7  public class Test {
8       public static void main(String[] args) {
9           List<String> list = new ArrayList<>();
10          list.add(0, "Array");
11          list.set(0, "List");
12
13          System.out.println(list);
14      }
15 }
```

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

- ◯ **[Array, List]**
- ◯ **An exception is thrown at runtime**
- ◯ **[Array]**
- ◯ **[List]** (Correct)
- ◯ **[List, Array]**

Explanation

list.add(0, "Array"); means list --> [Array],

list.set(0, "List"); means replace the current element at index 0 with the passed element "List". So after this operation, list --> [List]. In the console, [List] is printed.

Question 33: **Skipped**

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  public class Test {
5      public static void main(String[] args) {
6          final int i1 = 1;
7          final Integer i2 = 1;
8          final String s1 = ":ONE";
9
10         String str1 = i1 + s1;
11         String str2 = i2 + s1;
12
13         System.out.println(str1 == "1:ONE");
14         System.out.println(str2 == "1:ONE");
15     }
16 }
```

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

- ◯ **false**
  true

- ◯ **false**
  false

- ◯ **true**
  false (Correct)

- ◯ **true**
  true

Explanation

Please note that Strings computed by concatenation at compile time, will be referred by String Pool during execution. Compile time String concatenation happens when both of the operands are compile time constants, such as literal, final variable etc.

Whereas, Strings computed by concatenation at run time (if the resultant expression is not constant expression) are newly created and therefore distinct.

For the statement, `String str1 = i1 + s1;`, i1 is a final variable of int type and s1 is a final variable of String type. Hence, `i1 + s1` is a constant expression which is computed at compile-time and results in String literal "1:ONE".

This means during compilation, Java compiler translates the statement

```
String str1 = i1 + s1;
```

to

```
String str1 = "1:ONE";
```

As "1:ONE" is a String literal, hence at runtime it will be referred by String Pool.

On the other hand, for the statement, `String str2 = i2 + s1;`, `i2 + s1` is not a constant expression because i2 is neither of primitive type nor of String type, hence it is computed at run-time and returns a non-pool String object "1:ONE".

As, str1 refers to String Pool object "1:ONE", hence `str1 == "1:ONE"` returns true, whereas str2 refers to non-Pool String object "1:ONE" and hence `str2 == "1:ONE"` returns false.

Question 34: Skipped

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.util.ArrayList;
5  import java.util.List;
6
7  public class Test {
8       public static void main(String[] args) {
9           List<StringBuilder> dryFruits = new ArrayList<>();
10          dryFruits.add(new StringBuilder("Walnut"));
11          dryFruits.add(new StringBuilder("Apricot"));
12          dryFruits.add(new StringBuilder("Almond"));
13          dryFruits.add(new StringBuilder("Date"));
14
15          for(int i = 0; i < dryFruits.size(); i++)
16          {
17              if(i == 0) {
18                  dryFruits.remove(new StringBuilder("Almond"));
19              }
20           }
21
22          System.out.println(dryFruits);
23      }
24 }
```

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

- ◯ **[Walnut, Apricot, Date]**
- ◯ **[Walnut, Apricot, Almond, Date]** (Correct)
- ◯ **[Walnut, Date]**
- ◯ **An exception is thrown at runtime**

Explanation

In this example, code is trying to remove an item from the list while iterating using traditional for loop so one can think that this code would throw java.util.ConcurrentModificationException.

But note, java.util.ConcurrentModificationException will never be thrown for traditional for loop. It is thrown for for-each loop or while using Iterator/ListIterator.

In this case dryFruits.remove(new StringBuilder("Almond")); will never remove any items from the list as StringBuilder class doesn't override the equals(Object) method of Object class.

StringBuilder instances created at "dryFruits.add(new StringBuilder("Almond"));" and "dryFruits.remove(new StringBuilder("Almond"));" are at different memory locations and equals(Object) method returns false for these instances.

Question 37: Skipped

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

```
1  package com.udayan.oca;
2
3  import java.util.function.Predicate;
4
5  public class Test {
6      public static void main(String[] args) {
7          printNumbers(i -> i % 2 != 0);
8      }
9
10     private static void printNumbers(Predicate<Integer> predicate) {
11         for(int i = 1; i <= 10; i++) {
12             if(predicate.test(i)) {
13                 System.out.print(i);
14             }
15         }
16     }
17 }
```

- ◯ **12345678910**
- ◯ **1234567891011**
- ◯ **246810**
- ◯ **1357911**
- ◯ **13579** (Correct)

Explanation

In the boolean expression (predicate.test(i)): i is of primitive int type but auto-boxing feature converts it to Integer wrapper type.

test(Integer) method of Predicate returns true if passed number is an odd number, so given loop prints only odd numbers. for loops works for the numbers from 1 to 10.

Question 38: **Skipped**

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

```
1  //TestStudent.java
2  package com.udayan.oca;
3
4  class Student {
5       String name;
6       int age;
7       boolean result;
8       double height;
9  }
10
11 public class TestStudent {
12      public static void main(String[] args) {
13          Student stud = new Student();
14          System.out.println(stud.name + stud.height + stud.result + stud.age);
15      }
16 }
```

- ◯ **null0.0false0** (Correct)
- ◯ **null0false0**
- ◯ **null0.0true0**
- ◯ **null0.0ffalse0**

Explanation

name, height, result and age are instance variables of Student class. And instance variables are initialized to their respective default values.

name is initialized to null, age to 0, result to false and height to 0.0.

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

```
1  package com.udayan.oca;
2
3  public class Test {
4       static Boolean[] arr = new Boolean[1];
5       public static void main(String[] args) {
6           if(arr[0]) {
7               System.out.println(true);
8           } else {
9               System.out.println(false);
10          }
11      }
12 }
```

- ◯ **false**
- ◯ **true**
- ◯ **ArrayIndexOutOfBoundsException is thrown at runtime**
- ◯ **Compilation error**
- ◯ **NullPointerException is thrown at runtime** (Correct)

Explanation

All the array elements are initialized to their default values. arr is of Boolean type (reference type), so arr[0] is initialized to null.

if expression works with Boolean type variable, so "if(arr[0])" doesn't give compilation error but java runtime extracts the boolean value stored in arr[0] and it uses booleanValue() method.

arr[0].booleanValue() means booleanValue() method is invoked on null reference and hence NullPointerException is thrown at runtime.

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

```
1  package com.udayan.oca;
2
3  class Base {
4       public void m1() throws NullPointerException {
5           System.out.println("Base: m1()");
6       }
7  }
8
9  class Derived extends Base {
10      public void m1() throws RuntimeException {
11          System.out.println("Derived: m1()");
12      }
13 }
14
15 public class Test {
16      public static void main(String[] args) {
17          Base obj = new Derived();
18          obj.m1();
19      }
20 }
```

- ◯ **Derived: m1()** (Correct)
- ◯ **Compilation error in Test class**
- ◯ **Base: m1()**
- ◯ **Compilation error in Derived class**

Explanation

NullPointerException extends RuntimeException, but there are no overriding rules related to unchecked exceptions.

So, method m1() in Derived class correctly overrides Base class method.

Rest is simple polymorphism. obj refers to an instance of Derived class and hence obj.m1(); invokes method m1() of Derived class, which prints "Derived: m1()" to the console.

```
5
6   public class Test {
7       public static void main(String[] args) {
8           try {
9               System.out.println(1);
10          } catch (NullPointerException ex) {
11              System.out.println("ONE");
12          } catch (FileNotFoundException ex) {
13              System.out.println("TWO");
14          }
15          System.out.println("THREE");
16      }
17  }
```

- ◯ **None of the System.out.println statements are executed**
- ◯ **Compilation error** (Correct)
- ◯ **TWO**
  THREE
- ◯ **THREE**
- ◯ **ONE**
  THREE

Explanation

Java doesn't allow to catch specific checked exceptions if these are not thrown by the statements inside try block.

catch(FileNotFoundException ex) {} causes compilation error in this case as System.out.println(1); will never throw FileNotFoundException.

NOTE: Java allows to catch Exception type. catch(Exception ex) {} will never cause compilation error.

Question 45: Skipped

Consider following code snippet:

```
package com.udayan.test;

public class Exam {
     public static void main(String [] args) {
         System.out.println("All the best!");
     }
}
```

Location of Exam.java file:

```
D:.
└──WORK
    └──QUIZ
        └──SEC07
            ├──classes
            └──src
                └──com
                    └──udayan
                        └──test
                                Exam.java
```

You are currently at Sec07 folder.
D:\WORK\Quiz\Sec07>

Which of the following javac command, typed from above location, will generate Exam.class file structure under classes directory?

```
D:.
└──WORK
    └──QUIZ
        └──SEC07
            ├──classes
            |   └──com
            |       └──udayan
            |           └──test
            |                   Exam.class
            |
            └──src
                └──com
                    └──udayan
                        └──test
                                Exam.java
```

- ◯ **Not possible by javac command**
- ◯ **javac classes\ src\com\udayan\test\Exam.java**
- ◯ **javac -d classes\ Exam.java**
- ◯ **javac -d classes\ src\com\udayan\test\Exam.java** (Correct)

Explanation

Use -d option with javac command. As you are typing javac command from within Sec07 directory, hence path of java file relative to Sec07 directory needs to be given.

So, correct command is: javac -d classes\ src\com\udayan\test\Exam.java

Question 46: Skipped

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.time.LocalDate;
5
6  public class Test {
7       public static void main(String [] args) {
8           LocalDate date = LocalDate.of(2020, 9, 6);
9           System.out.println(date);
10      }
11 }
```

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

- ◯ **2020-09-06** (Correct)
- ◯ **2020-9-6**
- ◯ **2020-06-09**
- ◯ **2020-6-9**

Explanation

In LocalDate.of(int, int, int) method, 1st parameter is year, 2nd is month and 3rd is day of the month.

toString() method of LocalDate class prints the LocalDate object in ISO-8601 format: "uuuu-MM-dd".

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

```
1  package com.udayan.oca;
2
3  class Super {
4       public Super(int i) {
5           System.out.println(100);
6       }
7  }
8
9  class Sub extends Super {
10      public Sub() {
11          System.out.println(200);
12      }
13 }
14
15 public class Test {
16      public static void main(String[] args) {
17          new Sub();
18      }
19 }
```

- ◯ **Compilation Error** (Correct)
- ◯ **200**
  100
- ◯ **100**
  200
- ◯ **200**

Explanation

super(); is added by the compiler as the first statement in both the constructors.

Class Super extends from Object class and Object class has no-argument constructor, hence no issues with the constructor of Super class.

But no-arg constructor is not available in Super class, hence calling super(); from Sub class constructor gives compilation error.

Question 48: **Skipped**

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

```
1  package com.udayan.oca;
2
3  public class Test extends String {
4      @Override
5      public String toString() {
6          return "TEST";
7      }
8
9      public static void main(String[] args) {
10         Test obj = new Test();
11         System.out.println(obj);
12     }
13 }
```

- ◯ **Compilation error** (Correct)
- ◯ **TEST**
- ◯ **Exception is thrown at runtime**
- ◯ **Output string contains @ symbol**

Explanation

String is a final class so it cannot be extended.

Question 49: Skipped

Which of the following statement is correct for below code?

```
1  package com.udayan.oca;
2
3  public class Test {
4       public static void main(String[] args) {
5           final boolean flag = false;
6           while(flag) {
7               System.out.println("Good Morning!");
8           }
9       }
10 }
```

- ◯ **It will print "Good Morning!" once**
- ◯ **Compilation error** (Correct)
- ◯ **Infinite loop**
- ◯ **Program compiles and executes successfully but produces no output**

Explanation

final boolean flag = false; statement makes flag a compile time constant.

Compiler knows the value of flag, which is false at compile time and hence it gives "Unreachable Code" error.

Question 50: Skipped

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

```
1  package com.udayan.oca;
2
3  public class Test {
4      public static void main(String[] args) {
5          Boolean b = new Boolean("tRUe");
6          switch(b) {
7              case true:
8                  System.out.println("ONE");
9              case false:
10                 System.out.println("TWO");
11             default:
12                 System.out.println("THREE");
13         }
14     }
15 }
```

- ◯ **THREE**
- ◯ **ONE**
  TWO
  THREE
- ◯ **TWO**
  THREE
- ◯ **None of the other options** (Correct)

Explanation

switch can accept primitive types: byte, short, int, char; wrapper types: Byte, Short, Integer, Character; String and enums.

switch(b) causes compilation failure as b is of Boolean type.

Question 52: Skipped

```
1  package com.udayan.oca;
2
3  class TestException extends Exception {
4       public TestException() {
5           super();
6       }
7
8       public TestException(String s) {
9           super(s);
10      }
11 }
12
13 public class Test {
14      public void m1() throws _________ {
15          throw new TestException();
16      }
17 }
```

For the above code, fill in the blank with one option.

- ◯ **RuntimeException**
- ◯ **Object**
- ◯ **Error**
- ◯ **Exception** (Correct)

Explanation

Method m1() throws an instance of TestException, which is a checked exception as it extends Exception class.

So in throws clause we must provide:

1. Checked exception.

2. Exception of TestException type or it's super types (Exception, Throwable), Object cannot be used in throws clause.

```java
public class Test {
    public static void main(String[] args) {
        int a = 20;
        int var = --a * a++ + a-- - --a;
        System.out.println("a = " + a);
        System.out.println("var = " + var);
    }
}
```

- ◯ **Compilation error**
- ◯ **a = 25**
  var = 363
- ◯ **a = 18**
  var = 363 (Correct)
- ◯ **a = 363**
  var = 363

Explanation

int var = --a * a++ + a-- - --a;

int var = --a * (a++) + (a--) - --a;

int var = (--a) * (a++) + (a--) - (--a);

int var = ((--a) * (a++)) + (a--) - (--a);

int var = (((--a) * (a++)) + (a--)) - (--a);

int var = ((19 * (a++)) + (a--)) - (--a); //a = 19

int var = ((19 * 19) + (a--)) - (--a); //a = 20

int var = (361 + 20) - (--a); //a = 19

int var = 381 - (--a); //a = 19

int var = 381 - 18; //a = 18

int var = 363 // a = 18

So,

a = 18

Question 55: Skipped

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.util.ArrayList;
5  import java.util.List;
6
7  public class Test {
8      public static void main(String[] args) {
9          List<String> trafficLight = new ArrayList<>();
10         trafficLight.add(1, "RED");
11         trafficLight.add(2, "ORANGE");
12         trafficLight.add(3, "GREEN");
13
14         trafficLight.remove(new Integer(2));
15
16         System.out.println(trafficLight);
17     }
18 }
```

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

- ◯ **Compilation error**
- ◯ **[RED, ORANGE, GREEN]**
- ◯ **An exception is thrown at runtime** (Correct)
- ◯ **[RED, ORANGE]**
- ◯ **[RED, GREEN]**

Explanation

There is no element at index 0 so call to add element at index 1, "trafficLight.add(1, "RED");" throws an instance of java.lang.IndexOutOfBoundsException.

trafficLight.remove(new Integer(2)); matches with trafficLight.remove(Object) and hence no compilation error.

Question 62: Skipped

For the given code:

```
1  package com.udayan.oca;
2
3  interface I01 {
4      void m1();
5  }
6
7  public class Implementer extends Object implements I01{
8      protected void m1() {
9
10     }
11 }
```

- ◯ None of the other options.
- ◯ interface I01 gives compilation error as method m1 is not public.
- ◯ Method m1() in Implementer class is not implemented correctly. (Correct)
- ◯ Implementer class declaration is not correct.

Explanation

void m1(); in interface I01 is equivalent to public abstract void m1(); So method m1() is implicitly public and abstract.

In java, a class can extend from only one class but can implement multiple interfaces. Correct keywords are: extends and implements.

so class declaration is correct.

As method m1() is implicitly public in I01, hence overriding method in Implementer class should also be public. But it is protected and hence compiler complains.

Question 63: Skipped

Consider below code:

```
1  //Test.java
2  import java.time.LocalDate;
3  import java.time.format.DateTimeFormatter;
4
5  public class Test {
6      public static void main(String [] args) {
7          LocalDate date = LocalDate.of(1987, 9, 1);
8          String str = date.format(DateTimeFormatter.ISO_DATE_TIME);
9          System.out.println("Date is: " + str);
10     }
11 }
```

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

- ◯ **Date is: 1987-01-09**
- ◯ **Date is: 1987-09-01**
- ◯ **Date is: 01-09-1987**
- ◯ **Given code executes successfully but output does not match with the given options**
- ◯ **Runtime exception** (Correct)

Explanation

LocalDate object doesn't contain time part but ISO_DATE_TIME looks for time portion and throws exception at runtime.

For the OCA exam, you can check following DateTimeFormatter types: BASIC_ISO_DATE, ISO_DATE, ISO_LOCAL_DATE, ISO_TIME, ISO_LOCAL_TIME, ISO_DATE_TIME, ISO_LOCAL_DATE_TIME.

Question 66: Skipped

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.time.LocalDate;
5
6  public class Test {
7      public static void main(String [] args) {
8          LocalDate joiningDate = LocalDate.parse("2006-03-16");
9          System.out.println(joiningDate.withDayOfYear(29));
10     }
11 }
```

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

- ◯ **None of the other options**
- ◯ **2006-03-29**
- ◯ **2006-01-01**
- ◯ **2006-01-29** (Correct)

Explanation

joiningDate --> {2006-03-16}.

joiningDate.withDayOfYear(29) returns a new LocalDate object with the day of the Year altered.

A year has 365 days, so 29 means 29th day of the year, which is 29th Jan 2006.

NOTE: There are other with methods, you should know for the exam.
withDayOfMonth(int), withMonth(int) and withYear(int).

Question 67: Skipped

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.util.ArrayList;
5  import java.util.List;
6
7  public class Test {
8      public static void main(String[] args) {
9          List<Boolean> list = new ArrayList<>();
10         list.add(true);
11         list.add(new Boolean("tRue"));
12         list.add(new Boolean("abc"));
13
14         if(list.remove(1)) {
15             list.remove(1);
16         }
17
18         System.out.println(list);
19     }
20 }
```

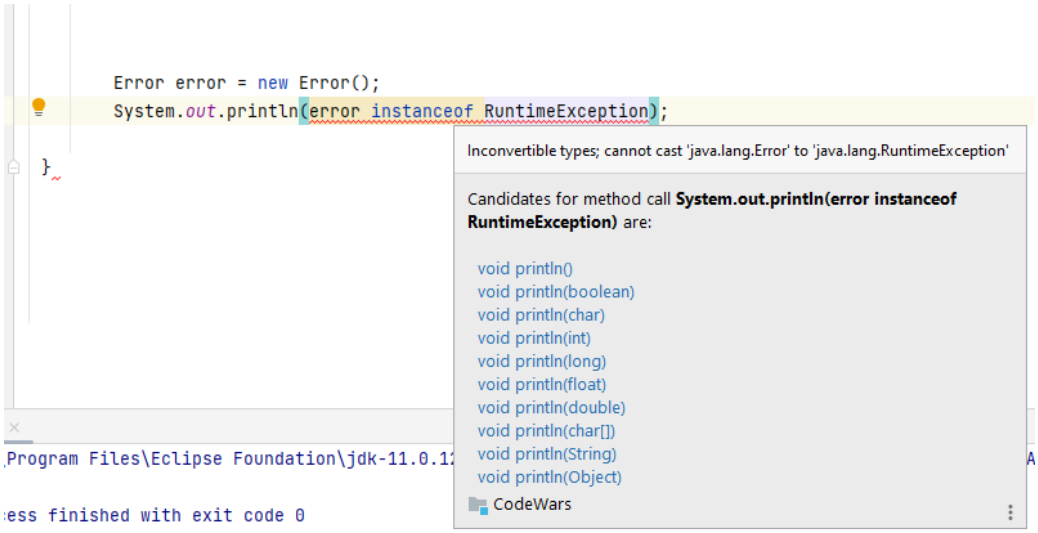
What will be the result of compiling and executing Test class?

- ◯ **[true, false]**
- ◯ **An exception is thrown at runtime**
- ◯ **[false]**
- ◯ **Compilation error**
- ◯ **[true]** (Correct)

Explanation

list.add(true); => Auto-boxing converts boolean literal true to Boolean instance

instanceOf should be a relation between two operand . parent child else below error at CE



Question 70: **Skipped**

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

```
1  package com.udayan.oca;
2
3  public class Test {
4      public static void main(String[] args) {
5          Error obj = new Error();
6          boolean flag1 = obj instanceof RuntimeException; //Line n1
7          boolean flag2 = obj instanceof Exception; //Line n2
8          boolean flag3 = obj instanceof Error; //Line n3
9          boolean flag4 = obj instanceof Throwable; //Line n4
10         System.out.println(flag1 + ":" + flag2 + ":" + flag3 + ":" + flag4);
11     }
12 }
```

- ◯ **false:false:true:true**
- ◯ **true:true:true:true**
- ◯ **Compilation error** (Correct)
- ◯ **false:true:true:true**
- ◯ **false:false:true:false**

Explanation

class Error extends Throwable, so `obj instanceof Error;` and `obj instanceof Throwable;` return true.

But Error class is not related to Exception and RuntimeException classes in multilevel inheritance and that is why Line n1 and Line n2 causes compilation error.

Question 4: **Incorrect**

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.time.LocalTime;
5
6  public class Test {
7      public static void main(String [] args) {
8          LocalTime time = LocalTime.of(23, 60);
9          System.out.println(time);
10     }
11 }
```

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

- ○ **00:01**
- ○ **An exception is thrown at runtime** (Correct)
- ○ **Compilation error**
- ◉ **23:60** (Incorrect)
- ○ **00:00**

Explanation

LocalTime.of(int hour, int minute) creates an instance of LocalTime class. Valid value for hour is: 0 to 23 and valid value for minute is 0 to 59.

java.time.DateTimeException is thrown if invalid values are passed as arguments.

NOTE: There are other overloaded of methods available:

LocalTime of(int hour, int minute, int second) and

LocalTime of(int hour, int minute, int second, int nanoOfSecond).

Valid value for second is: 0 to 59 and valid value for nano second is: 0 to 999,999,999.

String name= null;

Sop(name)-> null

Question 8: **Incorrect**

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

```
1  package com.udayan.oca;
2
3  public class Test {
4      public static void main(String[] args) {
5          /*INSERT*/
6          switch(var) {
7              case '7':
8                  System.out.println("Lucky no. 7");
9                  break;
10             default:
11                 System.out.println("DEFAULT");
12         }
13     }
14 }
```

- (●) **None of the other options** **(Incorrect)**
- ( ) `int var = 7;`
- ( ) `Integer var = 7;`
- ( ) `int var = '7';` **(Correct)**

Explanation

int var = 7; => DEFAULT,

Integer var = 7; => var is of Integer type and case contains char '7'. char '7' cannot be compared with Integer and hence compilation error. case '7' can easily be compared with int value but not with Integer type.

int var = '7'; => Lucky no. 7

HINT: There is no need to remember. case '7' value means you are trying to equate or compare var (Integer value) with '7' (char).

If assignment operation works then method invocation, switch expression parameter etc. will also work. Integer var = 7; is possible but Integer var = '7'; causes compilation error as char cannot be converted to Integer.

Question 9: **Correct**

```
1  public class Test {
2      private static int [] arr;
3      public static void main(String [] args) {
4          if(arr.length > 0 && arr != null) {
5              System.out.println(arr[0]);
6          }
7      }
8  }
```

Predict Output, if the above code is run with given command?

java Test

- ○ **No Output**
- ◉ **NullPointerException is thrown at runtime** **(Correct)**
- ○ **ArrayIndexOutOfBoundsException is thrown at runtime**
- ○ **Compilation error**

Explanation

Variable arr is a class variable of int [] type, so by default it is initialized to null.

In if block, arr.length > 0 is checked first. Accessing length property on null reference throws NullPointerException.

Correct logical if block declaration should be:

if(arr != null && arr.length > 0)

First check for null and then access properties/methods.

Question 10: **Incorrect**

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

```
1  package com.udayan.oca;
2
3  public class Test {
4       public static void main(String[] args) {
5           int [] arr1 = {1, 2, 3};
6           char [] arr2 = {'A', 'B'}; //ASCII code of 'A' is 65, 'B' is 66
7           arr1 = arr2;
8           for(int i = 0; i < arr1.length; i++) {
9               System.out.print(arr1[i] + " ");
10          }
11      }
12 }
```

- ○ **Compilation error** (Correct)
- ◉ **65 66** (Incorrect)
- ○ **A B**
- ○ **1 2 3**

Explanation

Initially arr1 refers to an int array object of 3 elements: 1, 2, 3

And arr2 refers to an char array object of 2 elements: 'A', 'B'.

Statement arr1 = arr2; gives compilation error as char [] is not compatible with int [] even though char is compatible with int.

```
class Test1 {

    static Double d1=null;
    //static int d2 = d1.intValue();

    public static void main(String ars[]) {


        double[] arr = {10.2, 25.5, 124.5};
        int[] iarr ={1,2,3};
        arr = iarr;


    }
}
```

Question 13: **Incorrect**

DateTimeFormatter is defined inside which package?

- ○ **java.util**
- ○ **java.text**
- ◉ **java.time** **(Incorrect)**
- ○ **java.time.format** **(Correct)**

Explanation

DateTimeFormatter is a part of "java.time.format" package, whereas LocalDate, LocalTime, LocalDateTime and Period are defined inside "java.time" package.

Question 14: **Correct**

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

```
1  package com.udayan.oca;
2
3  public class Test {
4      public static void main(String[] args) {
5          String [] arr = {"abc", "TrUe", "false", null, "FALSE"};
6          for(String s : arr) {
7              System.out.print(Boolean.valueOf(s) ? "T" : "F");
8          }
9      }
10 }
```

- ○ **TTTFT**
- ○ **NullPointerException is thrown at runtime**
- ◉ **FTFFF** **(Correct)**
- ○ **FFFFF**
- ○ **TTFTT**

Explanation

Boolean.valueOf(String s) returns true if passed String argument is not null and is equal, ignoring case, to the String "true". In all other cases it returns false.

Boolean.valueOf("abc") => false. As "abc".equalsIgnoreCase("true") is false.

Boolean.valueOf("TrUe") => true. As "TrUe".equalsIgnoreCase("true") is true.

Boolean.valueOf("false") => false. As "false".equalsIgnoreCase("true") is false.

Boolean.valueOf(null) => false. As passed argument is null.

Boolean.valueOf("FALSE") => false. As "FALSE".equalsIgnoreCase("true") is false.

Question 15: Skipped

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

```
1  package com.udayan.oca;
2
3  public class Test {
4       private static String s;
5       public static void main(String[] args) {
6           try {
7               System.out.println(s.length());
8           } catch(NullPointerException | RuntimeException ex) {
9               System.out.println("DONE");
10          }
11      }
12 }
```

- ◯ None of the above
- ◯ Executes successfully but no output
- ◯ Compilation error (Correct)
- ◯ DONE

Explanation

NullPointerException extends RuntimeException and in multi-catch syntax we can't specify multiple Exceptions related to each other in multilevel inheritance.

Sb.append(“”).length()->0

Question 17: **Incorrect**

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

```
1  package com.udayan.oca;
2
3  public class Test {
4      public static void print() {
5          System.out.println("static method");
6      }
7
8      public static void main(String[] args) {
9          Test obj = null;
10         obj.print();
11     }
12 }
```

- (●) **NullPointerException is thrown** (Incorrect)
- ( ) **None of the other options**
- ( ) **static method** (Correct)
- ( ) **Compilation error**

Explanation

print() is static method of class Test. So correct syntax to call method print() is Test.print();

but static methods can also be invoked using reference variable: obj.print(); Warning is displayed in this case.

Even though obj has null value, we don't get NullPointerException as objects are not needed to call static methods.

Question 19: **Incorrect**

Which of the following is the correct package declaration to declare Test class in com.exam.oca package?

- (●) `package com.exam.oca.Test;` **(Incorrect)**
- ( ) `package com.exam.oca;` **(Correct)**
- ( ) `package com.exam.oca.*;`
- ( ) `Package com.exam.oca;`

Explanation

To declare Test class in com.exam.oca package, use following declaration:

package com.exam.oca;

No wildcard (*) allowed in package declaration. Don't include class name in package declaration.

NOTE: all small case letters in package keyword.

NOTE: all small case letters in package keyword.

Question 20: **Incorrect**

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

```
1  package com.udayan.oca;
2
3  public class Test {
4       public static void main(String[] args) {
5           Boolean [] arr = new Boolean[2];
6           System.out.println(arr[0] + ":" + arr[1]);
7       }
8  }
```

- ◯ **true:true**
- ◉ **NullPointerException is thrown at runtime** (Incorrect)
- ◯ **false:false**
- ◯ **null:null** (Correct)

Explanation

Array elements are initialized to their default values.

arr is referring to an array of Boolean type, which is reference type and hence both the array elements are initialized to null and therefore in the output null:null is printed.

Question 21: **Incorrect**

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

```
1  package com.udayan.oca;
2
3  public class Test {
4       public static void main(String[] args) {
5           String [] arr = {"A", "B", "C", "D"};
6           arr[0] = arr[1];
7           arr[1] = "E";
8           for(String s : arr) {
9               System.out.print(s + " ");
10          }
11      }
12 }
```

- (●) **E E C D** (Incorrect)
- ( ) **Compilation error**
- ( ) **B E C D** (Correct)
- ( ) **A E C D**
- ( ) **An exception is thrown at runtime**

Question 23: **Incorrect**

Consider code of Test.java file:

```
1  package com.udayan.oca;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  public class Test {
7      public static void main(String[] args) {
8          List<Character> list = new ArrayList<>();
9          list.add(0, 'V');
10         list.add('T');
11         list.add(1, 'E');
12         list.add(3, 'O');
13
14         if(list.contains('O')) {
15             list.remove(3);
16         }
17
18         for(char ch : list) {
19             System.out.print(ch);
20         }
21     }
22 }
```

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

- ○ **VET** (Correct)
- ○ **VETO**
- ◉ **Runtime error** (Incorrect)
- ○ **Compilation error**
- ○ **VTE**

list.contains('O') => char 'O' is auto-boxed to Character object and as Character class overrides equals(String) method this expression returns true. Control goes inside if-block and executes: list.remove(3);.

list.remove(3); => Removes last element of the list. list --> [V,E,T].

for(char ch : list) => First list item is Character object, which is auto-unboxed and assigned to ch. This means in first iteration ch = 'V'; And after this it is simple enhanced for loop. Output is VET.

defined in LocalTime and LocalDateTime class. Hence obj.getHour() causes compilation failure.

Question 25: **Incorrect**

Consider below code:

```
1  package com.udayan.oca;
2
3  public class Test {
4       static Double d1;
5       int x = d1.intValue();
6
7       public static void main(String[] args) {
8           System.out.println("HELLO");
9       }
10 }
```

On execution, does Test class print "HELLO" on to the console?

- (●) **No, HELLO is not printed on to the console** (Incorrect)
- ( ) **Yes, HELLO is printed on to the console** (Correct)

Explanation

To invoke the special main method, JVM loads the class in the memory. At that time, static fields of Test class are initialized. d1 is of Double type so null is assigned to it.

x is not static variable, so int x = d1.intValue(); is not executed. Class is loaded successfully in the memory and "HELLO" is printed on to the console.

NOTE: new Test() will throw NullPointerException but not ExceptionInInitializerError.

Question 26: **Correct**

Given Code:

```
1  package com.udayan.oca;
2
3  import java.io.*;
4
5  class ReadTheFile {
6       static void print() { //Line 4
7           throw new IOException(); //Line 5
8       }
9  }
10
11 public class Test {
12      public static void main(String[] args) { //Line 10
13          ReadTheFile.print(); //Line 11
14          //Line 12
15      }
16 }
```

Which 2 changes are necessary so that code compiles successfully?

- [ ] **Replace Line 4 with** `static void print() throws Throwable {`

- [ ] **Surround Line 11 with below try-catch block:**

```
try {
     ReadTheFile.print();
} catch(IOException e) {
     e.printStackTrace();
}
```

- [ ] **Surround Line 11 with below try-catch block:**

```
try {
     ReadTheFile.print();
} catch(IOException | Exception e) {
     e.printStackTrace();
}
```

- [x] **Surround Line 11 with below try-catch block:** (Correct)

```
try {
     ReadTheFile.print();
} catch(Exception e) {
     e.printStackTrace();
}
```

- [x] **Replace Line 4 with** `static void print() throws Exception {` (Correct)

- [ ] **Replace Line 10 with** `public static void main(String[] args) throws IOException {`

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

```
1  package com.udayan.oca;
2
3  public class Test {
4      public static void main(String[] args) {
5          extractInt(2.7);
6          extractInt(2);
7      }
8
9      private static void extractInt(Double obj) {
10         System.out.println(obj.intValue());
11     }
12 }
```

- ○ **An exception is thrown at runtime**
- ○ **2**
  2
- ○ **3**
  2
- ○ **Compilation error in extractInt method**
- ◉ **Compilation error in main method** **(Correct)**

Explanation

extractInt method accepts argument of Double type.

extractInt(2.7); => 2.7 is double literal, so Java compiler would box it into Double type. At runtime obj.intValue() would print int portion of the Double data, which is 2.

extractInt(2); => Java compiler either does implicit casting or Wrapping but not both. 2 is int literal, Java compiler can't implicit cast it to double and then box it to Double. So this statement causes compilation failure.

Question 38: Skipped

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.time.LocalDate;
5  import java.time.format.DateTimeFormatter;
6
7  public class Test {
8       public static void main(String [] args) {
9           LocalDate date1 = LocalDate.parse("1947-08-15",
   DateTimeFormatter.ISO_DATE);
10          LocalDate date2 = LocalDate.parse("1947-08-15",
11
                                              DateTimeFormatter.ISO_LOCAL_DATE);
12          LocalDate date3 = LocalDate.of(1947, 8, 15);
13
14          System.out.println(date1.equals(date2) + " : " + date2.equals(date3));
15      }
16 }
```

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

- ◯ **Runtime exception**
- ◯ **false : false**
- ◯ **true : true** (Correct)
- ◯ **true : false**
- ◯ **false : true**

Explanation

ISO_LOCAL_DATE formatter formats the date without the offset, such as "1947-08-15".

ISO_DATE formatter formats the date with offset (if available), such as "1947-08-15" or "1947-08-15+05:30", but remember LocalDate object doesn't contain any offset information.

In this case, all the three date instances are meaningfully equal.

For the OCA exam, you can check following DateTimeFormatter types: BASIC_ISO_DATE, ISO_DATE, ISO_LOCAL_DATE, ISO_TIME, ISO_LOCAL_TIME, ISO_DATE_TIME, ISO_LOCAL_DATE_TIME.

Question 39: **Skipped**

How many String objects are there in the HEAP memory, when control is at Line 9?

```
1  package com.udayan.oca;
2
3  public class Test {
4       public static void main(String[] args) {
5           String s1 = new String("Java"); //Line 3
6           String s2 = "JaVa"; //Line 4
7           String s3 = "JaVa"; //Line 5
8           String s4 = "Java"; //Line 6
9           String s5 = "Java"; //Line 7
10
11          int i = 1; //Line 9
12
13      }
14 }
```

- ◯ **5**
- ◯ **2**
- ◯ **4**
- ◯ **3** **(Correct)**

Explanation

String s1 = new String("Java"); -> Creates 2 objects: 1 String Pool and 1 non-pool. s1 refers to non-pool object.

String s2 = "JaVa"; -> Creates 1 String pool object and s2 refers to it.

String s3 = "JaVa"; -> Doesn't create a new object, s3 refers to same String pool object referred by s2.

String s4 = "Java"; -> Doesn't create a new object, s4 refers to String Pool object created at Line 3.

String s5 = "Java"; -> Doesn't create a new object, s5 also refers to String Pool object created at Line 3.

So, at Line 9, 3 String objects are available in the HEAP memory: 2 String pool and 1 non-pool.

Question 40: **Incorrect**

Which of the following statement is correct for below code?

```
1  package com.udayan.oca;
2
3  public class Test {
4       public static void main(String[] args) {
5           final boolean flag;
6           flag = false;
7           while(flag) {
8               System.out.println("Good Morning!");
9           }
10      }
11 }
```

- (●) **Compilation error.** (Incorrect)
- ( ) **It will print "Good Morning!" once.**
- ( ) **Program compiles and executes successfully but produces no output.** (Correct)
- ( ) **Infinite loop.**

Explanation

final boolean flag; flag = false; doesn't make flag a compile time constant.

Compiler doesn't know flag's value at compile-time and hence it allows this syntax.

At runtime, as boolean expression of while loop is false, loop doesn't execute even once and hence no output.

Question 41: **Correct**

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


```
1  package com.udayan.oca;
2
3  public class Wall {
4       public static void main(String args[]) {
5           double area = 5.7;
6           String color;
7           if (area < 7)
8               color = "green";
9
10          System.out.println(color);
11      }
12 }
```

- ○ **null**
- ○ **green**
- ◉ **Compilation error** (Correct)
- ○ **NullPointerException**

Explanation

color is LOCAL variable and it must be initialized before it can be used.

As area is not compile time constant, java compiler doesn't have an idea of the value of variable area.

There is no else block available as well.

So compiler cannot be sure of whether variable color will be initialized or not, therefore System.out.println(color); causes compilation error.

Question 43: **Correct**

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.util.ArrayList;
5  import java.util.List;
6
7  public class Test {
8       public static void main(String[] args) {
9           Boolean [] arr = new Boolean[2];
10          List<Boolean> list = new ArrayList<>();
11          list.add(arr[0]);
12          list.add(arr[1]);
13
14          if(list.remove(0)) {
15              list.remove(1);
16          }
17
18          System.out.println(list);
19      }
20 }
```

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

- ◯ **Compilation error**
- ◉ **NullPointerException is thrown at runtime** (Correct)
- ◯ **[true]**
- ◯ **[]**
- ◯ **ArrayIndexOutOfBoundsException is thrown at runtime**
- ◯ **[false]**

```
2
3   import java.util.ArrayList;
4   import java.util.List;
5
6   abstract class Animal {}
7   class Dog extends Animal{}
8
9   public class Test {
10      public static void main(String [] args) {
11          List<Animal> list = new ArrayList<Dog>();
12          list.add(0, new Dog());
13          System.out.println(list.size() > 0);
14      }
15  }
```

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

- (●) **true** (Incorrect)
- ( ) **false**
- ( ) **Runtime exception**
- ( ) **Compilation error** (Correct)

Explanation

List is super type and ArrayList is sub type, hence List l = new ArrayList(); is valid syntax.

Animal is super type and Dog is sub type, hence Animal a = new Dog(); is valid syntax. Both depicts Polymorphism.

But in generics syntax, Parameterized types are not polymorphic, this means ArrayList<Animal> is not super type of ArrayList<Dog>. Remember this point. So below syntaxes are not allowed:

ArrayList<Animal> list = new ArrayList<Dog>(); OR List<Animal> list = new ArrayList<Dog>();

Question 47: **Incorrect**

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.time.LocalDate;
5
6  public class Test {
7       public static void main(String [] args) {
8           LocalDate date1 = LocalDate.parse("1980-03-16");
9           LocalDate date2 = LocalDate.parse("1980-03-16");
10          System.out.println(date1.equals(date2) + " : " + date1.isEqual(date2));
11      }
12 }
```

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

- ○ **true : true** (Correct)
- ○ **false : false**
- ○ **true : false**
- ◉ **false : true** (Incorrect)

Explanation

Both the methods "public boolean isEqual(ChronoLocalDate)" and "public boolean equals(Object)" return true if date objects are equal otherwise false.

NOTE: LocalDate implements ChronoLocalDate.

This option is is not available, hence correct answer is "None of the other options"

Question 50: **Incorrect**

Which of the following will give you current system time?

Select 2 options.

- [ ] `System.out.println(LocalDateTime.now());` **(Correct)**
- [x] `System.out.println(LocalTime.now());` **(Correct)**
- [ ] `System.out.println(new LocalTime());`
- [ ] `System.out.println(LocalDate.now());`
- [ ] `System.out.println(new LocalDate());`
- [ ] `System.out.println(new LocalDateTime());`

Explanation

new LocalDate(), new LocalTime() and new LocalDateTime() give compilation error as constructor of these classes are declared private.

System.out.println(LocalDate.now()); => Prints current date only.

System.out.println(LocalTime.now()); => Prints current time only.

System.out.println(LocalDateTime.now()); => Prints current date and time both.

Question 51: **Incorrect**

Consider below code of Test.java file:

```
1  package com.udayan.oca;
2
3  class Document {
4      int pages;
5      Document(int pages) {
6          this.pages = pages;
7      }
8  }
9
10 class Word extends Document {
11     String type;
12     Word(String type) {
13         super(20); //default pages
14         /*INSERT-1*/
15     }
16
17     Word(int pages, String type) {
18         /*INSERT-2*/
19         super.pages = pages;
20     }
21 }
22
23 public class Test {
24     public static void main(String[] args) {
25         Word obj = new Word(25, "TEXT");
26         System.out.println(obj.type + "," + obj.pages);
27     }
28 }
```

Currently above code causes compilation error.

Which of the options can successfully print TEXT,25 on to the console?

- ○ **Replace /*INSERT-1*/ with:**
  super.type = type;
  Replace /*INSERT-2*/ with:
  super(type);

- ○ **Replace /*INSERT-1*/ with:**
  super.type = type;
  Replace /*INSERT-2*/ with:
  this(type);

- ○ **Replace /*INSERT-1*/ with:**
  this(type);
  Replace /*INSERT-2*/ with:
  this.type = type;

- ○ **Replace /*INSERT-1*/ with:**
  this.type = type;
  Replace /*INSERT-2*/ with:
  this(type);
  (Correct)

- ◉ **None of the other options** **(Incorrect)**

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

public class Test {
    public static void main(String[] args) {
        String javaworld = "JavaWorld";
        String java = "Java";
        String world = "World";
        java += world;
        System.out.println(java == javaworld);
    }
}
```

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

- ○ **Java**
- ○ **JavaWorld**
- ○ **true**
- ○ **World**
- ◉ **false** **(Correct)**

Explanation

Please note that Strings computed by concatenation at compile time, will be referred by String Pool during execution. Compile time String concatenation happens when both of the operands are compile time constants, such as literal, final variable etc.

Whereas, Strings computed by concatenation at run time (if the resultant expression is not constant expression) are newly created and therefore distinct.

`java += world;` is same as `java = java + world;` and `java + world` is not a constant expression and hence is calculated at runtime and returns a non pool String object "JavaWorld", which is referred by variable 'java'.

On the other hand, variable 'javaworld' refers to String Pool object "JavaWorld". As both the variables 'java' and 'javaworld' refer to different String objects, hence `java == javaworld` returns false.

On the other hand, variable 'javaworld' refers to String Pool object "JavaWorld". As both the variables 'java' and 'javaworld' refer to different String objects, hence `java == javaworld` returns false.

Question 53: **Incorrect**

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.time.LocalDate;
5
6  public class Test {
7       public static void main(String [] args) {
8           LocalDate date = LocalDate.of(2068, 4, 15);
9           System.out.println(date.getMonth() + ":" + date.getMonthValue());
10      }
11 }
```

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

- ○ **April:3**
- ◉ **April:4** **(Incorrect)**
- ○ **APRIL:3**
- ○ **APRIL:4** **(Correct)**

Explanation

date.getMonth() returns the month of the year filed, using Month enum, all the enum constant names are in upper case.

date.getMonthValue() returns the value of the month.

NOTE: month value starts with 1 and it is different from java.util.Date API, where month value starts with 0.

Question 54: **Incorrect**

Which of the following statement is correct about below code?

```
1  package com.udayan.oca;
2
3  public class Test {
4       public static void main(String[] args) {
5           do {
6               System.out.println(100);
7           } while (true);
8
9           System.out.println("Bye");
10      }
11 }
```

- (●) **Compiles successfully and prints 100 in infinite loop** (Incorrect)
- ( ) **Unreachable code compilation error** (Correct)
- ( ) **Compiles successfully and prints "Bye"**
- ( ) **100**
  Bye

Explanation

Boolean expression of do-while loop uses literal true (compile-time constant), hence Java compiler knows that this loop is an infinte loop.

It also knows that once at runtime Java Control enters an infinite loop, none of the statements after loop block will get executed.

Hence it marks all the codes after infinite loop as Unreachable Code, which results in compilation error.

If boolean variable was used instead of boolean literal, then this program would have compiled and executed successfully.

```
1  public class DoWhileTest1 {
2      public static void main(String[] args) {
3          boolean flag = true;
4          do {
5              System.out.println(100);
6          } while (flag);
7
8          System.out.println("Bye");
9      }
10 }
```

Above program prints 100 in infinite loop and "Bye" never gets printed.

Question 55: **Incorrect**

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.util.ArrayList;
5  import java.util.List;
6
7  public class Test {
8       public static void main(String[] args) {
9           List<String> dryFruits = new ArrayList<>();
10          dryFruits.add("Walnut");
11          dryFruits.add("Apricot");
12          dryFruits.add("Almond");
13          dryFruits.add("Date");
14
15          for(String dryFruit : dryFruits) {
16              if(dryFruit.startsWith("A")) {
17                  dryFruits.remove(dryFruit);
18              }
19          }
20
21          System.out.println(dryFruits);
22      }
23 }
```

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

- (●) **[Walnut, Date]** (Incorrect)
- ( ) **Compilation error**
- ( ) **[Walnut, Apricot, Almond, Date]**
- ( ) **An exception is thrown at runtime** (Correct)

Explanation

ConcurrentModificationException exception may be thrown for following condition:

1. Collection is being iterated using Iterator/ListIterator or by using for-each loop.

And

2. Execution of Iterator.next(), Iterator.remove(), ListIterator.previous(), ListIterator.set(E) & ListIterator.add(E) methods. These methods may throw java.util.ConcurrentModificationException in case Collection had been modified by means other than the iterator itself, such as Collection.add(E) or Collection.remove(Object) or List.remove(int) etc.

PLEASE NOTE: for-each loop internally implements Iterator and invokes hasNext() and next() methods.

For the given code, 'dryFruits' list is being iterated using for-each loop (internally as an Iterator).

hasNext() method of Iterator has following implementation:

```
1  public boolean hasNext() {
2      return cursor != size;
3  }
```

Where cursor is the index of next element to return and initially it is 0.

1st Iteration: cursor = 0, size = 4, hasNext() returns true. iterator.next() increments the cursor by 1 and returns "Walnut".

2nd Iteration: cursor = 1, size = 4, hasNext() returns true. iterator.next() increments the cursor by 1 and returns "Apricot". As "Apricot" starts with "A", hence dryFruits.remove(dryFruit) removes "Apricot" from the list and hence reducing the list's size by 1, size becomes 3.

3rd Iteration: cursor = 2, size = 3, hasNext() returns true. iterator.next() method throws java.util.ConcurrentModificationException.

If you want to successfully remove the items from ArrayList, while using Iterator or ListIterator, then use Iterator.remove() or ListIterator.remove() method and NOT List.remove(...) method. Using List.remove(...) method while iterating the list (using the Iterator/ListIterator or for-each) may throw java.util.ConcurrentModificationException.

Question 57: **Incorrect**

Wrapper classes are defined in which of the following package?

- (●) **default package** **(Incorrect)**
- ( ) **java.util**
- ( ) **java.lang** **(Correct)**
- ( ) **java.io**

Explanation

All the wrapper classes are defined in java.lang package.

String and StringBuilder are also defined in java.lang package and that is why import statement is not required to use these classes.

Question 59: **Skipped**

Consider following code snippet:

```
1  package com.udayan.test;
2
3  public class Exam {
4      public static void main(String [] args) {
5          System.out.println("All the best!");
6      }
7  }
```

Location of files:

```
1  D:.
2  └───WORK
3      └───QUIZ
4          └───SEC07
5              ├───classes
6              |   └───com
7              |       └───udayan
8              |           └───test
9              |                   Exam.class
10             |
11             └───src
12                 └───com
13                     └───udayan
14                         └───test
15                                 Exam.java
```

You are currently at WORK folder.

D:\WORK>

Which of the following java command will show All the best! on to the console?

- ◯ **java -cp Quiz\Sec07\classes\ com.udayan.test.Exam** (Correct)
- ◯ **java Exam**
- ◯ **java -cp Quiz\Sec07\classes\com\udayan\test\ Exam**
- ◯ **java com.udayan.test.Exam**

Explanation

To execute Exam class from WORK folder, you should specify the classpath (Quiz\Sec07\classes\) which contains whole path of the class(com\udayan\test\Exam.class).

And you should also use fully qualified name of the class, which is com.udayan.test.Exam.

Hence correct option is: java -cp Quiz\Sec07\classes\ com.udayan.test.Exam

```
public class Test {
    public static void main(String[] args) {
        int i;
        outer:
        do {
            i = 5;
            inner:
            while (true) {
                System.out.println(i--);
                if (i == 4) {
                    break outer;
                }
            }
        } while (true);
    }
}
```

- ○ **5 3 2 1**
- ○ **Prints 5 once.** **(Correct)**
- ○ **Compilation error.**
- ○ **Prints 5 in an infinite loop.**

Explanation

"outer" and "inner" are valid label names.

On execution, control enters main method and creates int variable i.

On encountering do-while loop, control goes inside and initializes variable i to 5.

Then it executes while loop and it's boolean expression is always true.

System.out.println(i--); prints 5 to the console first, and then decrements the value of i by 1. So, i becomes 4.

Boolean expression of if(i == 4) evaluates to true. break outer; statement executes and takes the control out of do-while loop.

main method ends and program terminates successfully.

So, 5 gets printed only once.

Question 64: Skipped

Consider codes of 3 java files:

```
1  //Class1.java
2  package com.udayan.oca;
3
4  import java.io.FileNotFoundException;
5
6  public class Class1 {
7      public void read() throws FileNotFoundException {}
8  }
1  //Class2.java
2  package com.udayan.oca;
3
4  public class Class2 {
5      String Class2;
6      public void Class2() {}
7  }
1  //Class3.java
2  package com.udayan.oca;
3
4  public class Class3 {
5      private void print() {
6          private String msg = "HELLO";
7          System.out.println(msg);
8      }
9  }
```

Which of the following statement is true?

- ◯ **Only Class3.java compiles successfully**
- ◯ **Class1.java and Class2.java compile successfully** (Correct)
- ◯ **Class2.java and Class3.java compile successfully**
- ◯ **Class1.java and Class3.java compile successfully**
- ◯ **Only Class1.java compiles successfully**
- ◯ **Only Class2.java compiles successfully**

Question 66: Skipped

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

```
1  package com.udayan.oca;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  public class Test {
7      public static void main(String[] args) {
8          Integer i = 10;
9          List<Integer> list = new ArrayList<>();
10         list.add(i);
11         list.add(new Integer(i));
12         list.add(i);
13
14         list.removeIf(i -> i == 10);
15
16         System.out.println(list);
17     }
18 }
```

- ◯ **[]**
- ◯ **[10, 10, 10]**
- ◯ **[10]**
- ◯ **Compilation Error** (Correct)
- ◯ **[10, 10]**

Explanation

Variable "i" used in lambda expression clashes with another local variable "i" and hence causes compilation error.

Question 67: **Incorrect**

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.util.ArrayList;
5  import java.util.List;
6
7  public class Test {
8       public static void main(String[] args) {
9           List<String> list = new ArrayList<>(4);
10          list.add(0, "Array");
11          list.add(2, "List");
12
13          System.out.println(list);
14      }
15 }
```

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

- ○ **[Array, List]**
- ○ **[Array, null, List, null]**
- ○ **An exception is thrown at runtime** (Correct)
- ◉ **Compilation error** (Incorrect)

Explanation

ArrayList are different than arrays, though behind the scene ArrayList uses Object[] to store its elements.

There are 2 things related to ArrayList, one is capacity and another is actual elements stored in the list, returned by size() method. If you don't pass anything to the ArrayList constructor, then default capacity is 10 but this doesn't mean that an ArrayList instance will be created containing 10 elements and all will be initialized to null.

In fact, size() method will still return 0 for this list. This list still doesn't contain even a single element. You need to use add method or its overloaded counterpart to add items to the list. Even if you want to add null values, you should still invoke some methods, nothing happens automatically.

In this question, new ArrayList<>(4); creates an ArrayList instance which can initially store 4 elements but currently it doesn't store any data.

Another point you should remember for the certification exam: Addition of elements in ArrayList should be continuous. If you are using add(index, Element) method to add items to the list, then index should be continuous, you simply can't skip any index.

In this case, list.add(0, "Array"); adds "Array" to 0th index. so after this operation list --> [Array]. You can now add at 0th index (existing elements will be shifted right) or you can add at index 1 but not at index 2. list.add(2, "List"); throws an instance of java.lang.IndexOutOfBoundsException.

Question 68: **Correct**

Consider below code:

```
1  //Test.java
2  package com.udayan.oca;
3
4  import java.time.Period;
5
6  public class Test {
7       public static void main(String [] args) {
8           Period period = Period.of(0, 1000, 0);
9           System.out.println(period);
10      }
11 }
```

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

- ○ **p0y1000m0d**
- ○ **P0Y1000M0D**
- ◉ **P1000M** **(Correct)**
- ○ **p1000m**

Explanation

Check the toString() method of Period class. ZERO period is displayed as P0D, other than that, Period components (year, month, day) with 0 values are ignored.

toString()'s result starts with P, and for non-zero year, Y is appended; for non-zero month, M is appended; and for non-zero day, D is appended. P,Y,M and D are in upper case.

NOTE: Period.parse(CharSequence) method accepts the String parameter in "PnYnMnD" format, over here P,Y,M and D can be in any case.

Question 69: Skipped

Consider below code:

```
1  package com.udayan.oca;
2
3  public class Test {
4      public static void main(String[] args) {
5          StringBuilder sb = new StringBuilder();
6          try {
7              for(;;) {
8                  sb.append("OCA");
9              }
10         } catch(Exception e) {
11             System.out.println("Exception!!!");
12         }
13         System.out.println("Main ends!!!");
14     }
15 }
```

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

- ◯ **"Exception!!!" is printed on to the console and program terminates abruptly**
- ◯ **Program terminates abruptly** (Correct)
- ◯ **"Exception!!!" and "Main ends!!!" are printed on to the console and program terminates successfully**
- ◯ **"Main ends!!!" is printed on to the console and program terminates successfully**
- ◯ **"Exception!!!" is printed on to the console and program terminates successfully**

Explanation

for(;;) is an infinite loop and hence `sb.append("OCA");` causes OutOfMemoryError which is a subclass of Error class.

main(String []) method throws OutOfMemoryError and program terminates abruptly.