Top of Form

Question 1: **Correct**

Given code of Test.java file:

1. package com.udayankhattry.oca;
3. class X {
4. void A() {
5. System.out.print("A");
6. }
7. }
9. class Y extends X {
10. void A() {
11. System.out.print("A-");
12. }
14. void B() {
15. System.out.print("B-");
16. }
18. void C() {
19. System.out.print("C-");
20. }
21. }
23. public class Test {
24. public static void main(String[] args) {
25. X obj = new Y(); //Line n1
26. obj.A(); //Line n2
27. obj.B(); //Line n3
28. obj.C(); //Line n4
29. }
30. }

What will be the result of compiling and executing above code?

* 

**A-B-C-**

* 

**Compilation error in class Y**

* 

**Compilation error in class Test**

**(Correct)**

* 

**AB-C-**

**Explanation**

Class Y correctly extends class X and it overrides method A() and provides two new methods B() and C().

At Line n1, obj is of X type and therefore obj.B(); and obj.C(); cause compilation error as these methods are not defined in class X.

Bottom of Form

Top of Form

Question 2: **Correct**

Given code of Test.java file:

1. package com.udayankhattry.oca;
3. public class Test {
4. public static void main(String[] args) {
5. byte b1 = 10; //Line n1
6. int i1 = b1; //Line n2
7. byte b2 = i1; //Line n3
8. System.out.println(b1 + i1 + b2);
9. }
10. }

What is the result of compiling and executing Test class?

* 

**30 is printed on to the console**

* 

**Line n3 causes compilation error**

**(Correct)**

* 

**Line n1 causes compilation error**

* 

**Line n2 causes compilation error**

**Explanation**

Let us first check Line n1: byte b1 = 10;

Above statement compiles successfully, even though 10 is an int literal (32 bits) and b1 is of byte primitive type which can store only 8 bits of data.

Here java does some background task, if value of int literal can be easily fit to byte primitive type (-128 to 127), then int literal is implicitly casted to byte type.

So above statement is internally converted to:

byte b1 = (byte)10;

But if you specify any out of range value then it would not be allowed, e.g.

byte b = 128; // It would cause compilation failure as 128 is out of range value for byte type.

There is no issue with Line n2 as byte type (8 bits) can be easily assigned to int type (32 bits).

For line n3, `byte b2 = i1;`, expression on right hand side (i1) is neither a withing range literal value nor constant expression, hence it causes compilation failure.

To compile successfully, this expression needs to be explicitly casted, such as: `byte b2 = (byte)i1;`

Bottom of Form

Top of Form

Question 3: **Correct**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. public class Test {
4. public static void main(String[] args) {
5. String str = "PANIC";
6. StringBuilder sb = new StringBuilder("THET");
7. System.out.println(str.replace("N", sb)); //Line n1
8. }
9. }

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

* 

**PANIC**

* 

**Line n1 throws error at runtime**

* 

**Line n1 causes compilation error**

* 

**PATHETIC**

**(Correct)**

**Explanation**

String class has following two overloaded replace methods:

1. public String replace(char oldChar, char newChar) {}:

Returns a string resulting from replacing all occurrences of oldChar in this string with newChar. If no replacement is done, then source String object is returned. e.g.

"Java".replace('a', 'A') --> returns new String object "JAvA".

"Java".replace('a', 'a') --> returns the source String object "Java" (no change).

"Java".replace('m', 'M') --> returns the source String object "Java" (no change).

2. public String replace(CharSequence target, CharSequence replacement) {}:

Returns a new String object after replacing each substring of this string that matches the literal target sequence with the specified literal replacement sequence. e.g.

"Java".replace("a", "A") --> returns new String object "JAvA".

"Java".replace("a", "a") --> returns new String object "Java" (it replaces "a" with "a").

"Java".replace("m", "M") --> returns the source String object "Java" (no change).

As String, StringBuilder and StringBuffer all implement CharSequence, hence instances of these classes can be passed to replace method. Line n1 compiles successfully and on execution replaces "N" with "THET", and hence Line n1 prints PATHETIC on to the console.

Bottom of Form

Top of Form

Question 4: **Correct**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. public class Test {
4. public static void main(String[] args) {
5. StringBuilder sb = new StringBuilder("B"); //Line n1
6. sb.append(sb.append("A")); //Line n2
7. System.out.println(sb); //Line n3
8. }
9. }

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

* 

**ABA**

* 

**BA**

* 

**ABAB**

* 

**B**

* 

**AB**

* 

**ABBA**

* 

**BABA**

**(Correct)**

* 

**Compilation error at Line n2**

* 

**BAB**

**Explanation**

At Line n1:

sb --> {"B"}

append(...) method in StringBuilder class is overloaded to accept various arguments and 2 such arguments are String and CharSequence. It's return type is StringBuilder and as StringBuilder class implements CharSequence interface, hence 'sb.append("A")' can easily be passed as and argument to sb.append(...) method. Line n2 compiles successfully.

At Line n2:

sb.append(sb.append("A")); //sb --> {"B"}

sb.append({"BA"}); //sb --> {"BA"}

{"BABA"}

Hence, Line n3 prints BABA

Bottom of Form

Top of Form

Question 5: **Correct**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. abstract class Log {
4. abstract long count(); //Line n1
5. abstract Object get(); //Line n2
6. }
8. class CommunicationLog extends Log {
9. int count() { //Line n3
10. return 100;
11. }
13. String get() { //Line n4
14. return "COM-LOG";
15. }
16. }
18. public class Test {
19. public static void main(String[] args) {
20. Log log = new CommunicationLog(); //Line n5
21. System.out.print(log.count());
22. System.out.print(log.get());
23. }
24. }

Which of the following statement is correct?

* 

**Line n4 causes compilation error**

* 

**Line n5 causes compilation error**

* 

**Line n3 causes compilation error**

**(Correct)**

* 

**Given code compiles successfully and on execution prints 100COM-LOG on to the console**

**Explanation**

CommunicationLog class overrides count() and get() methods of Log class.

There are 2 rules related to return types:

1. If return type of overridden method is of primitive type, then overriding method should use same primitive type.

2. If return type of overridden method is of reference type, then overriding method can use same reference type or its sub-type (also known as covariant return type).

count() method at Line n1 returns long but overriding method at Line n3 returns int and that is why Line n3 causes compilation error.

get() method at Line n2 returns Object but overriding method at Line n4 returns String. String is a subclass of Object, so it is a case of covariant return type and hence allowed. Line n4 compiles successfully.

Bottom of Form

Top of Form

Question 6: **Correct**

Consider below code fragment:

1. String place = "MISSS";
2. System.out.println(place.replace("SS", "T"));

What is the output?

* 

**MIT**

* 

**MIST**

* 

**MISSS**

* 

**MITS**

**(Correct)**

**Explanation**

According to Javadoc, replace(CharSequence target, CharSequence replacement) method of String class returns a new String object after replacing each substring of this string that matches the literal target sequence with the specified literal replacement sequence. The replacement proceeds from the beginning of the string to the end, for example, replacing "aa" with "b" in the string "aaa" will result in "ba" rather than "ab".

"MISSS".replace("SS", "T"); returns "MITS".

Bottom of Form

Top of Form

Question 7: **Correct**

Given code of Test.java file:

1. package com.udayankhattry.oca;
3. class Base {
4. int id = 1000; //Line n1
6. Base() {
7. Base(); //Line n2
8. }
10. void Base() { //Line n3
11. System.out.println(++id); //Line n4
12. }
13. }
15. class Derived extends Base {
16. int id = 2000; //Line n5
18. Derived() {} //Line n6
20. void Base() { //Line n7
21. System.out.println(--id); //Line n8
22. }
23. }
25. public class Test {
26. public static void main(String[] args) {
27. Base base = new Derived(); //Line n9
28. }
29. }

What will be the result of compiling and executing above code?

* 

**0**

* 

**1001**

* 

**Compilation error**

* 

**1999**

* 

**2000**

* 

**-1**

**(Correct)**

* 

**An exception is thrown**

* 

**1000**

* 

**999**

* 

**2001**

**Explanation**

Method can have same name as that of the Class. Hence, void Base() is a valid method declaration in Base class.

Line n2 invokes the Base() method and not the constructor.

Subclass overrides the methods of superclass but it hides the variables of superclass.

Line n5 hides the variable created at Line n1, there is no rules related to hiding (type and access modifier can be changed).

Line n7 correctly overrides the Base() method of class Base.

Compiler adds super(); as the 1st statement inside the no-argument constructor of Base class and Derived class.

There is no compilation error, so let's check the execution.

new Derived() at Line n9 invokes the constructor of Derived class, at this point instance variable id is declared and 0 is assigned to it. In fact, instance variable id of Base class is also declared and 0 is assigned to it. Compiler added super(); as the first statement inside this constructor, hence control goes to the no-argument constructor of Base class.

Compiler added super(); as the first statement inside this constructor as well, hence it invokes the no-argument constructor of the Object class. No-argument constructor of Object class finishes its execution and control goes back to the constructor of Base class. Before it starts executing remaining statements inside the constructor, instance variable assignment statement (if available) are executed. This means 1000 is assigned to variable id of Base class.

Line n2 is executed next, Base() method defined in Derived class is executed. Which overriding method to invoke, is decided at runtime based on the instance. Instance is of Derived class (because of Line n9), hence control starts executing Base() method of Derived class.

Line n8 is executed next, Derived class hides the id variable of Base class and that is why at Line n8, id points to variable created at Line n5. This id variable still stores the value 0 as Base class's constructor has not finishes its execution.

value of id is decremented by 1, so id becomes -1 and -1 is printed on to the console. Base() method finishes its execution and control goes back to Line n2. No-argument constructor of Base class finishes its execution and control goes back to the constructor of Derived class. Before it starts executing remaining statements inside the constructor, instance variable assignment statement (if available) are executed. This means 2000 is assigned to variable id of Derived class.

No-argument constructor of Derived class finishes its execution and control goes back to Line n9. main(String []) method finishes its execution and program terminates successfully.

Hence, output is -1.

Bottom of Form

Top of Form

Question 8: **Correct**

Given code of Test.java file:

1. package com.udayankhattry.oca;
3. class Calculator {
4. int calculate(int i1, int i2) {
5. return i1 + i2;
6. }
8. double calculate(byte b1, byte b2) {
9. return b1 % b2;
10. }
11. }
13. public class Test {
14. public static void main(String[] args) {
15. byte b = 100;
16. int i = 20;
17. System.out.println(new Calculator().calculate(b, i));
18. }
19. }

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

* 

**5.0**

* 

**120**

**(Correct)**

* 

**5**

* 

**120.0**

* 

**Compilation error**

* 

**An exception is thrown at runtime**

**Explanation**

calculate method is correctly overloaded as both the methods have different signature: calculate(int, int) and calculate(byte, byte). Please note that there is no rule regarding return type for overloaded methods, return type can be same or different.

`new Calculator().calculate(b, i)` tags to `calculate(int, int)` as byte value is implicitly casted to int type.

Given code compiles successfully and on execution prints 120 on to the console.

Bottom of Form

Top of Form

Question 9: **Correct**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. public class Test {
4. public static void main(String[] args) {
5. int x = 10; //Line n1
6. if (false)
7. System.out.println(x); //Line n2
8. System.out.println("HELLO"); //Line n3
9. }
10. }

What is the result of compiling and executing Test class?

* 

**10**

**HELLO**

* 

**HELLO**

**(Correct)**

* 

**Compilation error at Line n1**

* 

**Compilation error at Line n2**

* 

**Compilation error at Line n3**

**Explanation**

Even though compiler is aware that Line n2 will never execute, but it doesn't tag it as unreachable code. Reason for this odd behavior is explained in the Java Language specification:

https://docs.oracle.com/javase/specs/jls/se8/html/jls-14.html#jls-14.21

Following statement results in a compile-time error:

while (false) { x=3; }

because the statement x=3; is not reachable; but the superficially similar case:

if (false) { x=3; }

does not result in a compile-time error. An optimizing compiler may realize that the statement x=3; will never be executed and may choose to omit the code for that statement from the generated class file, but the statement x=3; is not regarded as "unreachable" in the technical sense specified here.

The rationale for this differing treatment is to allow programmers to define "flag" variables such as:

static final boolean DEBUG = false;

and then write code such as:

if (DEBUG) { x=3; }

The idea is that it should be possible to change the value of DEBUG from false to true or from true to false and then compile the code correctly with no other changes to the program text.

Line n2 is not executed but Line n3 executes successfully and prints HELLO on to the console.

Bottom of Form

Top of Form

Question 10: **Correct**

Given code of Test.java file:

1. package com.udayankhattry.oca;
3. interface M {
4. public static void log() {
5. System.out.println("M");
6. }
7. }
9. abstract class A {
10. public static void log() {
11. System.out.println("N");
12. }
13. }
15. class MyClass extends A implements M {}
17. public class Test {
18. public static void main(String[] args) {
19. M obj1 = new MyClass();
20. obj1.log(); //Line n1
22. A obj2 = new MyClass();
23. obj2.log(); //Line n2
25. MyClass obj3 = new MyClass();
26. obj3.log(); //Line n3
27. }
28. }

Which of the following statements is correct?

* 

**Line n3 causes compilation error**

* 

**There is a compilation error in class A**

* 

**Given code compiles successfully**

* 

**Line n1 causes compilation error**

**(Correct)**

* 

**There is a compilation error in interface M**

* 

**Line n2 causes compilation error**

**Explanation**

As per Java 8, default and static methods were added in the interface. Interface M defines static method log(), there is no compilation error in interface M.

Also the scope of static log() method of M is limited to interface M and it can be invoked by using Interface name only, M.log().

Abstract class A also defines the static log() method. Abstract class can have 0 or more abstract methods. Hence, no compilation error in class A as well.

Super type reference variable can refer to an instance of Sub type, therefore the statement `M obj1 = new MyClass();` compiles successfully.

obj1 is of M type, hence `obj1.log();` tries to tag the static method of M but static log() method of M can only be invoked by using M.log();.

Therefore, Line n1 causes compilation error.

Scope of static log() method of A is not limited to class A only but MyClass also gets A.log() method in its scope.

There are different ways in which static method of an abstract class can be accessed:

1. By using the name of the abstract class: A.log(); //Preferred way

2. By using the reference variable of abstract class: A o1 = null; o1.log();

3. By using the name of the subclass: MyClass.log();

4. By using the reference variable of the subclass: MyClass o2 = null; o2.log();

Hence, Line n2 and Line n3 compile successfully.

Bottom of Form

Top of Form

Question 11: **Correct**

Given code of Test.java file:

1. package com.udayankhattry.oca;
3. class Parent {
4. String quote = "MONEY DOESN'T GROW ON TREES";
5. }
7. class Child extends Parent {
8. String quote = "LIVE LIFE KING SIZE";
9. }
11. class GrandChild extends Child {
12. String quote = "PLAY PLAY PLAY";
13. }
15. public class Test {
16. public static void main(String[] args) {
17. GrandChild gc = new GrandChild();
18. System.out.println(/\*INSERT\*/);
19. }
20. }

Which of the following options, if used to replace /\*INSERT\*/, will compile successfully and on execution will print MONEY DOESN'T GROW ON TREES on to the console?

Select 2 options.

* 

**gc.quote**

* 

**((Parent)gc).quote**

**(Correct)**

* 

**((Parent)(Child)gc).quote**

**(Correct)**

* 

**(Parent)(Child)gc.quote**

* 

**(Parent)gc.quote**

**Explanation**

As instance variables are hidden by subclasses and not overridden, therefore instance variable can be accessed by using explicit casting.

Let's check all the options one by one:

gc.quote => It refers to "PLAY PLAY PLAY" as gc is of GrandChild class.

(Parent)gc.quote => gc.quote will be evaluated first as dot (.) operator has higher precedence than cast. gc.quote refers to String, hence it cannot be casted to Parent type. This would cause compilation error.

((Parent)gc).quote => Variable 'gc' is casted to Parent type, so this expression refers to "MONEY DOESN'T GROW ON TREES". It is one of the correct options.

((Parent)(Child)gc).quote => 'gc' is of GrandChild type, it is first casted to Child and then to Parent type and finally quote variable is accessed, so this expression refers to "MONEY DOESN'T GROW ON TREES". It is also one of the correct options.

(Parent)(Child)gc.quote => gc.quote will be evaluated first as dot (.) operator has higher precedence than cast. gc.quote refers to String, hence it cannot be casted to Child type. This would cause compilation error.

Bottom of Form

Top of Form

Question 12: **Correct**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. public class Test {
4. public static void main(String[] args) {
5. int ctr = 100;
6. one: for (int i = 0; i < 10; i++) {
7. two: for (int j = 0; j < 7; j++) {
8. three: while (true) {
9. ctr++;
10. if (i > j) {
11. break one;
12. } else if (i == j) {
13. break two;
14. } else {
15. break three;
16. }
17. }
18. }
19. }
20. System.out.println(ctr);
21. }
22. }

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

* 

**106**

* 

**100**

* 

**Compilation error**

* 

**101**

* 

**104**

* 

**105**

* 

**103**

* 

**102**

**(Correct)**

**Explanation**

For the 1st loop variable 'i' infers to int type, so no issues for 1st loop and for the 2nd loop variable 'j' infers to int type, so no issues for 2nd loop as well.

Let's check the iteration:

1st iteration of loop one: i = 0

    1st iteration of loop two: j = 0

        1st iteration of loop three: ctr = 101. As `i == j` evaluates to true, hence `break two;` gets executed, which takes the control out of loop two and hence to the increment expression (i++) of loop one.

2nd iteration of loop one; i = 1

    1st iteration of loop two: j = 0

        1st iteration of loop three; ctr = 102. As `i > j` evaluates to true, hence `break one;` gets executed, which takes the control out of the loop one.

`System.out.println(ctr);` prints 102 on to the console.

Bottom of Form

Top of Form

Question 13: **Correct**

Given code of Thought.java file:

1. public class Thought {
2. /\*INSERT\*/ {
3. System.out.println("All is well");
4. }
5. }

Which 3 options, if used to replace /\*INSERT\*/, will compile successfully and on execution will print "All is well" on to the console?

* 

**public static Void main(String [] args)**

* 

**static public void Main(String [] args)**

* 

**public static void main(String... a)**

**(Correct)**

* 

**public void main(String... args)**

* 

**public void static main(String [] args)**

* 

**static public void main(String [] args)**

**(Correct)**

* 

**protected static void main(String [] args)**

* 

**public static void main(String [] a)**

**(Correct)**

**Explanation**

As System.out.println needs to be executed on executing the Test class, this means special main method should replace /\*INSERT\*/.

Special main method's name should be "main" (all characters in lower case), should be static, should have public access specifier and it accepts argument of String [] type (Varargs syntax String... can also be used). String [] argument can use any identifier name, even though in most of the cases you will see "args" is used. Position of static and public can be changed but return type 'void' must come just before the method name.

Let's check all the given options one by one:

public void static main(String [] args): Compilation error as return type 'void' must come just before the method name 'main'.

protected static void main(String [] args): Compiles successfully but as this method is not public, hence an Error regarding missing main method is thrown on execution.

public void main(String... args): Compiles successfully but as this method is not static, hence an Error regarding non-static main method is thrown on execution.

static public void Main(String [] args): Compiles successfully but as 'M' is capital in method 'Main', hence it is not special main method. An Error regarding missing main method is thrown on execution.

static public void main(String [] args): Valid definition, it compiles successfully and on execution prints "All is well" on to the console.

public static void main(String [] a): Valid definition, it compiles successfully and on execution prints "All is well" on to the console.

public static Void main(String [] args): Compilation error as Void is a final class in Java and in this case compiler expects main method to return a value of Void type. If you add `return null;` to the main method code will compile successfully but on execution an Error will be thrown mentioning that return type must be 'void' ('v' in lower-case).

public static void main(String... a): Valid definition, it compiles successfully and on execution prints "All is well" on to the console.

Bottom of Form

Top of Form

Question 14: **Correct**

Consider incomplete code of M.java file

1. class M {
2. }
4. \_\_\_\_\_\_\_\_ class N {
5. }

Following options are available to fill the above blank:

1. public

2. private

3. protected

4. final

5. abstract

How many above options can be used to fill above blank (separately and not together) such that there is no compilation error?

* 

**All five options**

* 

**Only one option**

* 

**Only four options**

* 

**Only two options**

**(Correct)**

* 

**Only three options**

**Explanation**

Top-level class can use only two access modifiers [public and default(don't specify anything)]. private and protected cannot be used.

As file name is M.java, hence class N cannot be public.

Top-level class can be final, hence it is a correct option.

Top-level class can be abstract and hence it is also a correct option.

Bottom of Form

Top of Form

Question 15: **Correct**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. public class Test {
4. public static void main(String[] args) {
5. char c1 = 'a'; //ASCII code of 'a' is 97
6. int i1 = c1; //Line n1
7. System.out.println(i1); //Line n2
8. }
9. }

What is the result of compiling and executing Test class?

* 

**Line n1 causes compilation failure**

* 

**97**

**(Correct)**

* 

**a**

* 

**Line n1 causes runtime error**

**Explanation**

Range of char data type is from 0 to 65535 and hence it can be easily assigned to int type. println() method is overloaded to accept char type and int type both. If char type value is passed, it prints char value and if int type value is passed, it prints int value.

As i1 is of int type, hence corresponding int value, which is 97, is printed on to the console.

Bottom of Form

Top of Form

Question 16: **Correct**

Given code of Test.java file:

1. package com.udayankhattry.oca;
3. public class Test {
4. public static void main(String[] args) {
5. System.out.println(new RuntimeException()); //Line n1
6. System.out.println(new RuntimeException("HELLO")); //Line n2
7. System.out.println(new RuntimeException(new RuntimeException("HELLO"))); //Line n3
8. }
9. }

Does above code compile successfully?

* 

**Yes**

**(Correct)**

* 

**No**

**Explanation**

Throwable is the root class of the exception hierarchy and it contains some useful constructors:

1. public Throwable() {...} : No-argument constructor

2. public Throwable(String message) {...} : Pass the detail message

3. public Throwable(String message, Throwable cause) {...} : Pass the detail message and the cause

4. public Throwable(Throwable cause) {...} : Pass the cause

Exception and RuntimeException classes also provide similar constructors.

Hence all 3 statements Line n1, Line n2 and Line n3 compile successfully.

Throwable class also contains methods, which are inherited by all the subclasses (Exception, RuntimeException etc.)

1. public String getMessage() {...} : Returns the detail message (E.g. detail message set by 2nd and 3rd constructor)

2. public String toString() {} :

Returns a short description of this throwable. The result is the concatenation of:

the name of the class of this object

": " (a colon and a space)

the result of invoking this object's getLocalizedMessage() method

If getLocalizedMessage returns null, then just the class name is returned.

Because of the toString() method,

Line n1 prints "java.lang.RuntimeException".

Line n2 prints "java.lang.RuntimeException: HELLO"

Line n3 prints "java.lang.Exception: java.lang.RuntimeException: HELLO"

Bottom of Form

Top of Form

Question 17: **Correct**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. public class Test {
4. public static void main(String [] args) {
5. boolean status = true;
6. System.out.println(status = false || status = true | status = false);
7. System.out.println(status);
8. }
9. }

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

* 

**false**

**true**

* 

**true**

**false**

* 

**true**

**true**

* 

**false**

**false**

* 

**Compilation error**

**(Correct)**

**Explanation**

System.out.println(status = false || status = true | status = false);

As it contains multiple operators, hence let's group the operators first.

System.out.println(status = false || status = (true | status) = false); //Bitwise inclusive OR | has highest precedence over logical or || and assignment =

For assignment operator to work, left operand must be variable but in above case, `(true | status) = false` causes compilation failure as left operand (true | status) evaluates to a boolean value and not boolean variable.

Bottom of Form

Top of Form

Question 18: **Correct**

Given code of Test.java file:

1. package com.udayankhattry.oca;
3. public class Test {
4. private static void div(int i, int j) {
5. try {
6. System.out.println(i / j);
7. } catch(ArithmeticException e) {
8. throw (RuntimeException)e;
9. }
10. }
12. public static void main(String[] args) {
13. try {
14. div(5, 0);
15. } catch(ArithmeticException e) {
16. System.out.println("AE");
17. } catch(RuntimeException e) {
18. System.out.println("RE");
19. }
20. }
21. }

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

* 

**AE is printed on to the console and program terminates successfully**

**(Correct)**

* 

**Compilation error**

* 

**Program ends abruptly**

* 

**RE is printed on to the console and program terminates successfully**

**Explanation**

Any RuntimeException can be thrown without any need it to be declared in throws clause of surrounding method.

`throw (RuntimeException)e;` doesn't cause any compilation error.

Even though variable 'e' is type casted to RuntimeException but exception object is still of ArithmeticException, which is caught in main method and 'AE' is printed to the console.

Bottom of Form

Top of Form

Question 19: **Correct**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. public class Test {
4. public static void main(String [] args) {
5. int num = 10;
6. if(num++ == num++) {
7. System.out.println("EQUAL " + num);
8. } else {
9. System.out.println("NOT EQUAL " + num);
10. }
11. }
12. }

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

* 

**NOT EQUAL 12**

**(Correct)**

* 

**EQUAL 11**

* 

**NOT EQUAL 11**

* 

**EQUAL 12**

**Explanation**

Given boolean expression:

(num++ == num++) //num=10

(10 == num++) //Left side operand is evaluated first, value 10 is used in the expression and variable num is incremented by 1, so num=11

(10 == 11) //Right side operand is evaluated next, value 11 is used in the expression and variable num is incremented by 1, so num = 12

Above expression evaluates to false, hence else block is executed and NOT EQUAL 12 is printed on to the console.

Bottom of Form

Top of Form

Question 20: **Correct**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. class Counter {
4. static int ctr = 0;
5. int count = 0;
6. }
8. public class Test {
9. public static void main(String[] args) {
10. Counter ctr1 = new Counter();
11. Counter ctr2 = new Counter();
12. Counter ctr3 = new Counter();
14. for(int i = 1; i <= 5; i++ ) {
15. ctr1.ctr++;
16. ctr1.count++;
17. ctr2.ctr++;
18. ctr2.count++;
19. ctr3.ctr++;
20. ctr3.count++;
21. }
23. System.out.println(ctr3.ctr + ":" + ctr3.count);
24. }
25. }

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

* 

**5:5 is printed on to the console**

* 

**Compilation error**

* 

**15:15 is printed on to the console**

* 

**15:5 is printed on to the console**

**(Correct)**

**Explanation**

Each instance of the class contains separate copies of instance variable and share one copy of static variable.

There are 3 instances of Counter class created by main method and these are referred by ctr1, ctr2 and ctr3.

As 'ctr' is a static variable of Counter class, hence ctr1.ctr, ctr2.ctr and ctr3.ctr refer to the same variable. In fact, 'Counter.ctr' is the preferred way to refer the static variable 'ctr' but ctr1.ctr, ctr2.ctr and ctr3.ctr are also allowed.

As 'count' is an instance variable, so there are 3 separate copies: ctr1.count, ctr2.count, ctr3.count.

On the completion of for loop: ctr1.count = 5, ctr2.count = 5 and ctr3.count = 5 and Counter.ctr = 15.

15:5 is printed on to the console.

Bottom of Form

Top of Form

Question 21: **Correct**

Below is the code of Test.java file:

1. package com.udayankhattry.oca;
3. import java.util.ArrayList;
4. import java.util.List;
6. public class Test {
7. public static void main(String[] args) {
8. List<String> places = new ArrayList<>();
9. places.add("Austin");
10. places.add("Okinawa");
11. places.add("Giza");
12. places.add("Manila");
13. places.add("Batam");
14. places.add("Giza");
16. if(places.remove("Giza"))
17. places.remove("Austin");
19. System.out.println(places);
20. }
21. }

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

* 

**[Austin, Okinawa, Manila, Batam]**

* 

**Compilation error**

* 

**[Okinawa, Manila, Batam, Giza]**

**(Correct)**

* 

**An exception is thrown at runtime**

* 

**[Austin, Okinawa, Giza, Manila, Batam, Giza]**

* 

**[Austin, Okinawa, Manila, Batam, Giza]**

* 

**[Okinawa, Manila, Batam]**

* 

**[Okinawa, Giza, Manila, Batam]**

**Explanation**

remove(Object) method of List interface removes the first occurrence of the specified element from the list, if it is present. If this list does not contain the element, it is unchanged. remove(Object) method returns true, if removal was successful otherwise false. Initially list has: [Austin, Okinawa, Giza, Manila, Batam, Giza]. places.remove("Giza") removes the first occurrence of "Giza" and after the successful removal, list has: [Austin, Okinawa, Manila, Batam, Giza]. places.remove("Giza") returns true, control goes inside if block and executes places.remove("Austin"); places list contains "Austin", so after the removal list has: [Okinawa, Manila, Batam, Giza].

Bottom of Form

Top of Form

Question 22: **Correct**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. public class Test {
4. public static void main(String[] args) {
5. for(int x = 10, y = 11, z = 12; y > x && z > y; y++, z -= 2) {
6. System.out.println(x + y + z);
7. }
8. }
9. }

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

* 

**33**

**32**

* 

**Compilation error**

* 

**33**

**(Correct)**

* 

**32**

* 

**34**

**Explanation**

Basic/Regular for loop has following form:

for ( [ForInit] ; [Expression] ; [ForUpdate] ) {...}

[ForInit] can be local variable initialization or the following expressions:

Assignment

PreIncrementExpression

PreDecrementExpression

PostIncrementExpression

PostDecrementExpression

MethodInvocation

ClassInstanceCreationExpression

[ForUpdate] can be following expressions:

Assignment

PreIncrementExpression

PreDecrementExpression

PostIncrementExpression

PostDecrementExpression

MethodInvocation

ClassInstanceCreationExpression

The [Expression] must have type boolean or Boolean, or a compile-time error occurs. If [Expression] is left blank, it evaluates to true.

All the expressions can be left blank; for(;;) is a valid for loop and it is an infinite loop as [Expression] is blank and evaluates to true.

Multiple comma separated statements are allowed for [ForInit] and [ForUpdate] expressions, where as [Expression] must be single expression which results in boolean or Boolean.

In the given for loop:

[ForInit] = int x = 10, y = 11, z = 12: It is allowed. 3 variables are declared and initialized. x = 10, y = 11 & z = 12.

[Expression] = y > x && z > y = (y > x) && (z > y) [Relational operator has higher precedence than logical AND]. This expression is valid and results in boolean value.

[ForUpdate] = y++, z -= 2. It is allowed. y is incremented by 1 and z is decremented by 2.

Let's check the loop's iteration:

1st iteration: x = 10, y = 11, z = 12. (y > x) && (z > y) = (11 > 10) && (12 > 11) = true && true = true. Loop's body is executed and prints x + y + z = 10 + 11 + 12 = 33 on to the console.

2nd iteration: [ForUpdate] is executed. y = 12, z = 10. (y > x) && (z > y) = (12 > 10) && (10 > 12) = true && false = false.

Control goes out of for loop and program terminates successfully.

Loop's body executes once and prints 33 on to the console.

Bottom of Form

Top of Form

Question 23: **Correct**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. class Super {
4. Super() {
5. System.out.print("Reach");
6. }
7. }
9. class Sub extends Super {
10. Sub() {
11. Super();
12. System.out.print("Out");
13. }
14. }
16. public class Test {
17. public static void main(String[] args) {
18. new Sub();
19. }
20. }

What will be the result of compiling and executing above code?

* 

**It prints ReachOut on to the console**

* 

**It prints OutReach on to the console**

* 

**Compilation Error in Sub class**

**(Correct)**

* 

**Compilation Error in Test class**

* 

**Compilation Error in Super class**

**Explanation**

Parent (Super) class constructor is invoked by `super();` (all letters in lowercase) from within the constructor of subclass.

First statement inside no-argument constructor of Sub class is: `Super();` (Letter 'S' is in uppercase) and hence it causes compilation error.

Bottom of Form

Top of Form

Question 24: **Correct**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. public class Test {
4. public static void main(String [] args) {
5. String text = "RISE ";
6. text = text + (text = "ABOVE ");
7. System.out.println(text);
8. }
9. }

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

* 

**RISE ABOVE RISE**

* 

**RISE ABOVE**

**(Correct)**

* 

**ABOVE ABOVE**

* 

**RISE RISE ABOVE**

**Explanation**

Initially text refers to "RISE ".

Given statement:

text = text + (text = "ABOVE ");

text = "RISE " + (text = "ABOVE "); //Left operand of + operator is evaluated first, text --> "RISE "

text = "RISE " + "ABOVE "; //Right operand of + operator is evaluated next, text --> "ABOVE "

text = "RISE ABOVE "; //text --> "RISE ABOVE "

Hence `System.out.println(text);` prints 'RISE ABOVE ' on to the console.

Bottom of Form

Top of Form

Question 25: **Correct**

Consider below code of Test.java file:

1. public class Test {
2. public static void main(String [] args) {
3. System.out.println("String");
4. }
6. public static void main(Integer [] args) {
7. System.out.println("Integer");
8. }
10. public static void main(byte [] args) {
11. System.out.println("byte");
12. }
13. }

And the commands:

javac Test.java

java Test 10

What is the result?

* 

**String**

**(Correct)**

* 

**An Exception is thrown at runtime**

* 

**Compilation error**

* 

**byte**

* 

**Integer**

**Explanation**

Like any other method, main method can also be overloaded. But main method called by JVM is always with String [] parameter. Don't get confused with 10 as it is passed as "10".

Execute above class with any command line arguments or 0 command line argument, output will always be "String".

Bottom of Form

Top of Form

Question 26: **Correct**

Given code of Test.java file:

1. package com.udayankhattry.oca;
3. import java.io.FileNotFoundException;
4. import java.io.IOException;
6. class Base {
7. Base() throws IOException {
8. System.out.print(1);
9. }
10. }
12. class Derived extends Base {
13. Derived() throws FileNotFoundException {
14. System.out.print(2);
15. }
16. }
18. public class Test {
19. public static void main(String[] args) throws Exception {
20. new Derived();
21. }
22. }

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

* 

**Test class executes successfully and prints 12 on to the console**

* 

**Compilation error in both Base and Derived classes**

* 

**Compilation error only in Base class**

* 

**Test class executes successfully and prints 21 on to the console**

* 

**Compilation error only in Derived class**

**(Correct)**

**Explanation**

It is legal for the constructors to have throws clause.

Constructors are not inherited by the Derived class so there is no method overriding rules related to the constructors but as one constructor invokes other constructors implicitly or explicitly by using this(...) or super(...), hence exception handling becomes interesting.

Java compiler adds super(); as the first statement inside Derived class's constructor:

Derived() throws FileNotFoundException {

    super(); //added by the compiler

    System.out.print(2);

}

As super(); invokes the constructor of Base class (which declares to throw IOException), compiler complains as Derived class no-argument constructor doesn't declare to throw IOException. It declares to throw FileNotFoundException (subclass of IOException), which is not enough for the instances of IOException.

Bottom of Form

Top of Form

Question 27: **Correct**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. import java.time.LocalDate;
5. public class Test {
6. public static void main(String [] args) {
7. LocalDate date = LocalDate.parse("2020-08-31");
8. System.out.println(date.plusMonths(-6));
9. }
10. }

What is the result?

* 

**Compilation error**

* 

**2020-02-30**

* 

**2020-02-29**

**(Correct)**

* 

**2020-02-28**

* 

**An exception is thrown at runtime**

* 

**2020-02-31**

**Explanation**

plusMonths(long) method of LocalDate class returns a copy of this LocalDate with the specified number of months added. Negative argument will subtract the passed month(s), hence date.plusMonths(-6) doesn't cause any compilation error.

This method adds the specified amount to the months field in three steps:

    Add the input months to the month-of-year field

    Check if the resulting date would be invalid

    Adjust the day-of-month to the last valid day if necessary

For the given code,

2020-08-31 plus -6 months would subtract 6 months from the given date and would result in the invalid date 2020-02-31. Instead of returning an invalid result, the last valid day of the month, 2020-02-29, is returned.

Please note, 2020 is leap year and hence last day of February is 29 and not 28.

Bottom of Form

Top of Form

Question 28: **Correct**

Given code of Test.java file:

1. class A {
2. public static void main(String [] args) {
3. System.out.println("A");
4. }
5. }
7. class B {
8. public static void main(String [] args) {
9. System.out.println("B");
10. }
11. }
13. class C {
14. public static void main(String [] args) {
15. System.out.println("C");
16. }
17. }
19. class D {
20. public static void main(String [] args) {
21. System.out.println("D");
22. }
23. }

Which of the following options is correct?

* 

**To print C on to the console, execute below commands:**

**javac C.java**

**java C**

* 

**To print C on to the console, execute below commands:**

**javac Test.java**

**java C**

**(Correct)**

* 

**Test.java file is not a valid java file as it doesn't contain code for class Test**

* 

**Test.java file will compile successfully but expected output is not possible**

* 

**To print C on to the console, execute below commands:**

**javac Test.java**

**java Test**

**Explanation**

Test.java is a valid java file. As none of the classes in Test.java file are public, hence file name can use any valid Java identifier.

As file name is Test.java, hence to compile the code below command is used:

javac Test.java

Execution of above command creates 4 class files: A.class, B.class, C.class & D.class.

To print C on to the console, class C must be executed. To execute C class, command is:

java C

Bottom of Form

Top of Form

Question 29: **Correct**

Consider below codes of 2 java files:

1. //Flyable.java
2. package com.udayankhattry.oca;
4. public interface Flyable {
5. static int horizontalDegree() { //Line n1
6. return 20;
7. }
9. default void fly() {
10. System.out.println("Flying at " + horizontalDegree() + " degrees."); //Line n2
11. }
13. void land();
14. }
15. //Aeroplane.java
16. package com.udayankhattry.oca;
18. public class Aeroplane implements Flyable {
19. public void land() {
20. System.out.println("Landing at " + -Flyable.horizontalDegree() + " degrees."); //Line n3
21. }
23. public static void main(String[] args) {
24. new Aeroplane().fly();
25. new Aeroplane().land();
26. }
27. }

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

* 

**Compilation error at Line n2**

* 

**Compilation error at Line n3**

* 

**Compilation error at Line n1**

* 

**Given code compiles successfully and on execution prints below in the output:**

**Flying at 20 degrees.**

**Landing at -20 degrees.**

**(Correct)**

**Explanation**

As per Java 8, default and static methods were added in the interface and default methods can invoke static method as well. Hence, there is no issue with the Flyable interface.

class Aeroplane implements Flyable interface, hence it inherits the default method fly() and static method horizontalDegree() can be accessed using Flyable.horizontalDegree(). It also provides the implementation of land() method. There is no issue with Aeroplane class as well.

On execution below text is printed on to the console:

Flying at 20 degrees.

Landing at -20 degrees.

Bottom of Form

Top of Form

Question 30: **Correct**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. interface I1 {
4. public static void print(String str) {
5. System.out.println("I1:" + str.toUpperCase());
6. }
7. }
9. class C1 implements I1 {
10. void print(String str) {
11. System.out.println("C1:" + str.toLowerCase());
12. }
13. }
15. public class Test {
16. public static void main(String[] args) {
17. I1 obj = new C1();
18. obj.print("Java");
19. }
20. }

Which of the following statements is correct?

* 

**Given code compiles successfully and on execution prints I1:JAVA on to the console**

* 

**Given code compiles successfully and on execution prints C1:java on to the console**

* 

**Class C1 causes compilation error**

* 

**Interface I1 causes compilation error**

* 

**Class Test causes compilation error**

**(Correct)**

**Explanation**

As per Java 8, default and static methods were added in the interface. Interface I1 defines static method print(String), there is no compilation error in interface I1.

Also the scope of print(String) method of I1 is limited to interface I1 and it can be invoked by using Interface name only, I1.print("").

class C1 implements I1 and it also defines print(String) instance method. Even though class C1 implements I1, it doesn't have static print(String) method in its scope, therefore class C1 compiles successfully.

Super type reference variable can refer to an instance of Sub type, therefore the statement `I1 obj = new C1();` compiles successfully.

obj is of I1 type, hence `obj.print("Java");` tries to tag the static method of I1 but static print(String) method of I1 can only be invoked by using I1.print("Java");.

Therefore, `obj.print("Java");` causes compilation error.

Bottom of Form

Top of Form

Question 31: **Correct**

Given code of Test.java file:

1. package com.udayankhattry.oca;
3. class Paper {
4. static String getType() { //Line n1
5. return "GENERIC";
6. }
7. }
9. class RuledPaper extends Paper {
10. String getType() { //Line n2
11. return "RULED";
12. }
13. }
15. public class Test {
16. public static void main(String[] args) {
17. Paper paper = new RuledPaper(); //Line n3
18. System.out.println(paper.getType()); //Line n4
19. }
20. }

Which of the following statements is true for above code?

* 

**Code compiles successfully and on execution prints GENERIC on to the console**

* 

**Compilation error in RuledPaper class**

**(Correct)**

* 

**Compilation error in Test class**

* 

**Code compiles successfully and on execution prints RULED on to the console**

**Explanation**

Instance method of subclass cannot override the static method of superclass.

Instance method at Line n2 tries to override the static method at Line n1 and hence Line n2 causes compilation error.

There is no issue with Line n3 as reference variable of superclass can refer to an instance of subclass.

At Line n4, paper.getType() doesn't cause compilation error but as this syntax creates confusion, so it is not a good practice to access the static variables or static methods using reference variable, instead class name should be used. Paper.getType() is the preferred syntax.

Bottom of Form

Top of Form

Question 32: **Incorrect**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. interface Perishable1 {
4. default int maxDays() {
5. return 1;
6. }
7. }
9. interface Perishable2 extends Perishable1 {
10. default int maxDays() {
11. return 2;
12. }
13. }
15. class Milk implements Perishable2, Perishable1 {}
17. public class Test {
18. public static void main(String[] args) {
19. Perishable1 obj = new Milk();
20. System.out.println(obj.maxDays());
21. }
22. }

Which of the following statements is correct?

* 

**Interface Perishable2 causes compilation error**

* 

**Given code compiles successfully and on execution Test class prints 2 on to the console**

**(Correct)**

* 

**Class Test causes compilation error**

* 

**Given code compiles successfully and on execution Test class prints 1 on to the console**

* 

**Class Milk causes compilation error**

**(Incorrect)**

**Explanation**

As per Java 8, default methods were added in the interface. Interface Perishable1 defines default method maxDays(), there is no compilation error in interface Perishable1. Method maxDays() is implicitly public in Perishable1.

interface Perishable2 extends Perishable1 and it overrides the default method maxDays() of Document, overriding method in Perishable2 is implicitly public. Interface Perishable2 compiles successfully.

Class Milk implements Perishable2 and Perishable1. Although it is redundant for Milk class to implement Preishable1 as Perishable2 already extends Perishable1.

There is no conflict in Milk class as it inherits the default method maxDays() of Perishable2 interface. Milk class compiles successfully.

`Perishable1 obj = new Milk();` It compiles fine as Perishable1 is supertype and Milk is subtype.

`obj.maxDays()` executes the default maxDays() method of Perishable2 interface and it returns 2. `System.out.println(obj.maxDays());` prints 2 on to the console.

Bottom of Form

Top of Form

Question 33: **Correct**

Consider below code snippet:

1. int i = 10;
2. System.out.println(i > 3 != false);

What is the result?

* 

**null**

* 

**false**

* 

**true**

**(Correct)**

* 

**Compilation error**

**Explanation**

Given Expression:

i > 3 != false

It has 2 operators > and !=. > has higher precedence over !=, hence given expression can be written as:

(i > 3) != false

Let's solve above expression:

true != false

true

Hence true is printed on to the console.

Bottom of Form

Top of Form

Question 34: **Incorrect**

Given code of Test.java file:

1. package com.udayankhattry.oca;
3. interface Rideable {
4. void ride(String name);
5. }
7. class Animal {}
9. class Horse extends Animal implements Rideable {
10. public void ride(String name) {
11. System.out.println(name.toUpperCase() + " IS RIDING THE HORSE");
12. }
13. }
15. public class Test {
16. public static void main(String[] args) {
17. Animal horse = new Horse();
18. /\*INSERT\*/
19. }
20. }

Which of the following options, if used to replace /\*INSERT\*/, will compile successfully and on execution will print EMMA IS RIDING THE HORSE on to the console?

Select 4 options.

* 

**horse.ride("EMMA");**

**(Incorrect)**

* 

**((Rideable)(Horse)horse).ride("EMMA");**

**(Correct)**

* 

**((Horse)(Rideable)horse).ride("emma");**

**(Correct)**

* 

**(Rideable)horse.ride("emma");**

* 

**(Horse)horse.ride("EMMA");**

* 

**(Horse)(Rideable)horse.ride("EMMA");**

* 

**((Rideable)horse).ride("emma");**

**(Correct)**

* 

**(Rideable)(Horse)horse.ride("EMMA");**

* 

**((Horse)horse).ride("Emma");**

**(Correct)**

**Explanation**

Let's check all the options one by one:

horse.ride("EMMA"); ✗ Variable 'horse' is of Animal type and ride(String) method is not defined in Animal class, therefore it causes compilation error.

(Horse)horse.ride("EMMA"); ✗ horse.ride("EMMA") will be evaluated first as dot (.) operator has higher precedence than cast. horse.ride("EMMA") returns void, hence it cannot be casted to Horse type. This would cause compilation error.

((Horse)horse).ride("Emma"); ✓ Variable 'horse' refers to an instance of Horse type and variable 'horse' is casted to Horse type. Horse class has ride(String) method, hence no compilation error. ride(String) method of Horse class will get invoked at runtime and will print the expected output. As, name.toUpperCase() method is invoked, hence it doesn't matter in what case you pass the name, in the output name will always be displayed in the upper case.

(Rideable)horse.ride("emma"); ✗ horse.ride("EMMA") will be evaluated first as dot (.) operator has higher precedence than cast. horse.ride("EMMA") returns void, hence it cannot be casted to Rideable type. This would cause compilation error.

((Rideable)horse).ride("emma"); ✓ Variable 'horse' refers to an instance of Horse type and variable 'horse' is casted to Rideable type (super type of Horse). Rideable interface has ride(String) method, hence no compilation error. ride(String) method of Horse class will get invoked at runtime and will print the expected output.

(Rideable)(Horse)horse.ride("EMMA"); ✗ horse.ride("EMMA") will be evaluated first as dot (.) operator has higher precedence than cast. horse.ride("EMMA") returns void, hence it cannot be casted to Horse type. This would cause compilation error.

(Horse)(Rideable)horse.ride("EMMA"); ✗ horse.ride("EMMA") will be evaluated first as dot (.) operator has higher precedence than cast. horse.ride("EMMA") returns void, hence it cannot be casted to Rideable type. This would cause compilation error.

((Rideable)(Horse)horse).ride("EMMA"); ✓ Variable 'horse' refers to an instance of Horse type, it is first casted to Horse type and then casted to Rideable type. Rideable interface has ride(String) method, hence no compilation error. ride(String) method of Horse class will get invoked at runtime and will print the expected output.

((Horse)(Rideable)horse).ride("emma"); ✓ Variable 'horse' refers to an instance of Horse type, it is first casted to Rideable type and then casted to Horse type. Horse class has ride(String) method, hence no compilation error. ride(String) method of Horse class will get invoked at runtime and will print the expected output.

Bottom of Form

Top of Form

Question 35: **Correct**

Consider below codes of 3 java files:

1. //Buyable.java
2. package com.udayankhattry.oca;
4. public interface Buyable {
5. int salePercentage = 85;
7. public static String salePercentage() {
8. return salePercentage + "%";
9. }
10. }
11. //Book.java
12. package com.udayankhattry.oca;
14. public class Book implements Buyable {}
15. //Test.java
16. package com.udayankhattry.oca;
18. public class Test {
19. public static void main(String[] args) {
20. Buyable [] arr = new Buyable[2];
21. for(Buyable b : arr) {
22. System.out.println(b.salePercentage); //Line n1
23. System.out.println(b.salePercentage()); //Line n2
24. }
26. Book [] books = new Book[2];
27. for(Book b : books) {
28. System.out.println(b.salePercentage); //Line n3
29. System.out.println(b.salePercentage()); //Line n4
30. }
31. }
32. }

Which of the following statements are correct?

Select 2 options.

* 

**There is a compilation error at Line n1**

* 

**There is a compilation error at Line n3**

* 

**There is a compilation error at Line n2**

**(Correct)**

* 

**There is a compilation error in Book.java file**

* 

**There is a compilation error in Buyable.java file**

* 

**There is a compilation error at Line n4**

**(Correct)**

**Explanation**

Variable 'salePercentage' declared inside interface Buyable is implicitly public, static and final. As per Java 8, default and static methods were added in the interface. There is no compilation error in Buyable.java file.

class Book implements Buyable interface but as there is no abstract method in Buyable interface, hence Book class is not needed to implement any method. Book.java file compiles successfully.

`Buyable [] arr = new Buyable[2];` creates one dimensional array of 2 elements of Buyable type and both the elements are initialized to null.

There are some difference in which static variables and static methods of the interface are accessed.

Correct and only way to access static method of an Interface is by using the name of the interface, such as Buyable.salePercentage(). Line n2 and Line n4 cause compilation error.

As far as public static final variable of interface is concerned, even through the correct way to access static variable is by using the name of the interface, such as Buyable.salePercentage but it can also be accessed by using following:

Reference variable of the interface: Buyable obj1 = null; System.out.println(obj1.salePercentage);

Name of the implementer class: System.out.println(Book.salePercentage);

Reference variable of the implementer class: Book obj2 = null; System.out.println(obj2.salePercentage);

Hence, Line n1 and Line n3 compile successfully.

Bottom of Form

Top of Form

Question 36: **Correct**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. public class Test {
4. public static void main(String[] args) {
5. String arr1 [], arr2, arr3 = null; //Line n1
6. arr1 = new String[2];
7. arr1[0] = "A";
8. arr1[1] = "B";
9. arr2 = arr3 = arr1; //Line n2
10. log(arr2); //Line n3
11. }
13. private static void log(String... vals) {
14. for(String s : vals)
15. System.out.print(s);
16. }
17. }

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

* 

**It executes successfully and prints B on to the console**

* 

**Line n2 causes compilation error**

**(Correct)**

* 

**It executes successfully and prints BA on to the console**

* 

**Line n3 causes compilation error**

* 

**It executes successfully and prints A on to the console**

* 

**Line n1 causes compilation error**

* 

**It executes successfully and prints AB on to the console**

**Explanation**

arr1 is of String[] type, where as arr2 and arr3 are of String type. As all three arr1, arr2 and arr3 are of reference type, hence null can be assigned to all these variables. Line n1 compiles successfully.

Statement at Line n2: arr2 = arr3 = arr1;

=> arr2 = (arr3 = arr1); //assignment operator is right to left associative.

arr3 is of String type and arr1 is of String [] type, hence (arr3 = arr1) causes compilation error.

Though you had to select one correct option, hence no need to look further but I am providing explanation for Line n3 as well.

log(String...) method can be called using a String [] or a String instance or mutliple String instances:

log(new String[] {"A", "B"});

log("A");

log("A", "B");

As arr2 is of String type, hence `log(arr2);` (Line n3) compiles successfully.

Bottom of Form

Top of Form

Question 37: **Correct**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. class MyClass {
4. MyClass() {
5. System.out.println(101);
6. }
7. }
9. class MySubClass extends MyClass {
10. final MySubClass() {
11. System.out.println(202);
12. }
13. }
14. public class Test {
15. public static void main(String[] args) {
16. System.out.println(new MySubClass());
17. }
18. }

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

* 

**202**

**101**

**<Some text containing @ symbol>**

* 

**202**

**<Some text containing @ symbol>**

* 

**101**

**202**

**<Some text containing @ symbol>**

* 

**101**

**<Some text containing @ symbol>**

* 

**Compilation error**

**(Correct)**

**Explanation**

Constructors cannot use final, abstract or static modifiers. As no-argument constructor of MySubClass uses final modifier, therefore it causes compilation error.

Bottom of Form

Top of Form

Question 38: **Correct**

Given code of Test.java file:

1. package com.udayankhattry.oca;
3. public class Test {
4. public static void convert(String s)
5. throws IllegalArgumentException, RuntimeException, Exception {
6. if(s.length() == 0) {
7. throw new RuntimeException("LENGTH SHOULD BE GREATER THAN 0");
8. }
9. }
10. public static void main(String [] args) {
11. try {
12. convert("");
13. }
14. catch(IllegalArgumentException | RuntimeException | Exception e) { //Line 14
15. System.out.println(e.getMessage()); //Line 15
16. } //Line 16
17. catch(Exception e) {
18. e.printStackTrace();
19. }
20. }
21. }

Line 14 causes compilation error. Which of the following changes enables to code to print LENGTH SHOULD BE GREATER THAN 0?

* 

**Comment out Line 14, Line 15 and Line 16**

* 

**Replace Line 14 with 'catch(IllegalArgumentException | RuntimeException e) {'**

* 

**Replace Line 14 with 'catch(RuntimeException | Exception e) {'**

* 

**Replace Line 14 with 'catch(RuntimeException e) {'**

**(Correct)**

* 

**Replace Line 14 with 'catch(IllegalArgumentException | Exception e) {'**

**Explanation**

Throwable is the root class of the exception hierarchy and it contains some useful constructors:

1. public Throwable() {...} : No-argument constructor

2. public Throwable(String message) {...} : Pass the detail message

3. public Throwable(String message, Throwable cause) {...} : Pass the detail message and the cause

4. public Throwable(Throwable cause) {...} : Pass the cause

Exception and RuntimeException classes also provide similar constructors.

Throwable class also contains methods, which are inherited by all the subclasses (Exception, RuntimeException etc.)

1. public String getMessage() {...} : Returns the detail message (E.g. detail message set by 2nd and 3rd constructor)

2. public String toString() {} :

Returns a short description of this throwable. The result is the concatenation of:

the name of the class of this object

": " (a colon and a space)

the result of invoking this object's getLocalizedMessage() method

If getLocalizedMessage() returns null, then just the class name is returned.

In multi-catch statement, classes with multi-level hierarchical relationship can't be used.

RuntimeException is subclass of Exception, IllegalArgumentException is indirect subclass of Exception and IllegalArgumentException is subclass of RuntimeException, hence these pairs can't be used in multi-catch statement.

Only one option is left to replace Line 14 with 'catch(RuntimeException e) {'.

Commenting out Line 14, Line 15 and Line 16 will resolve the compilation error but it will print the whole stack trace rather than just printing the message.

Bottom of Form

Top of Form

Question 39: **Correct**

Given code of Test.java file:

1. package com.udayankhattry.oca;
3. interface Document {
4. default String getType() {
5. return "TEXT";
6. }
7. }
9. interface WordDocument extends Document {
10. String getType();
11. }
13. class Word implements WordDocument {}
15. public class Test {
16. public static void main(String[] args) {
17. Document doc = new Word(); //Line n1
18. System.out.println(doc.getType()); //Line n2
19. }
20. }

Which of the following statements is correct?

* 

**Class Word causes compilation error**

**(Correct)**

* 

**Test class compiles successfully and on execution prints TEXT on to the console**

* 

**Interface Document causes compilation error**

* 

**Interface WordDocument causes compilation error**

**Explanation**

As per Java 8, default methods were added in the interface. Interface Document defines default method getType(), there is no compilation error in interface Document. Method getType() is implicitly public in Document.

interface WordDocument extends Document and it overrides the default method getType() of Document, overriding method in WordDocument is implicitly abstract and public. An interface in java can override the default method of super type with abstract modifier. interface WordDocument compiles successfully.

class Word implements WordDocument and as WordDocument interface has abstract method getType(), and as class Word doesn't implement the getType() method hence it causes compilation failure.

Bottom of Form

Top of Form

Question 40: **Correct**

Given code of Test.java file:

1. package com.udayankhattry.oca;
3. import java.io.FileNotFoundException;
5. public class Test {
6. public static void main(String[] args) {
7. try {
8. System.out.println(args[1].length());
9. } catch (RuntimeException ex) {
10. System.out.println("ONE");
11. } catch (FileNotFoundException ex) {
12. System.out.println("TWO");
13. }
14. System.out.println("THREE");
15. }
16. }

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

* 

**Compilation error**

**(Correct)**

* 

**None of the System.out.println statements is executed**

* 

**ONE**

**THREE**

* 

**TWO**

**THREE**

* 

**THREE**

**Explanation**

java.io.FileNotFoundException exception is a checked exception.

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.

Bottom of Form

Top of Form

Question 41: **Correct**

Consider below code fragment:

private void emp() {}

And the statements:

1. Given code compiles successfully if it is used inside the class named 'emp'

2. Given code compiles successfully if it is used inside the class named 'Emp'

3. Given code compiles successfully if it is used inside the class named 'employee'

4. Given code compiles successfully if it is used inside the class named 'Employee'

5. Given code compiles successfully if it is used inside the class named 'Student'

6. Given code compiles successfully if it is used inside the class named '\_emp\_'

How many statements are true?

* 

**Two statements**

* 

**Four statements**

* 

**Only one statement**

* 

**All six statements**

**(Correct)**

* 

**Three statements**

* 

**Five statements**

**Explanation**

`private void emp() {}` is a valid method declaration.

Class name and method name can be same and that is why given method can be declared in any of the given classes: 'emp', 'Emp', 'employee', 'Employee', 'Student' and '\_emp\_'.

'\_emp\_' is also a valid Java identifier.

Bottom of Form

Top of Form

Question 42: **Correct**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. import java.util.ArrayList;
4. import java.util.List;
6. public class Test {
7. public static void main(String[] args) {
8. List<String> list;
9. list = new ArrayList<>(); //Line n1
10. list.add("A");
11. list.add("E");
12. list.add("I");
13. list.add("O");
14. list.add("U");
15. list.addAll(list.subList(0, 4)); //Line n2
16. System.out.println(list);
17. }
18. }

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

* 

**Line n2 causes compilation error**

* 

**[A, E, I, O, U, A, E, I, O]**

**(Correct)**

* 

**An exception is thrown at runtime by Line n2**

* 

**[A, E, I, O, U]**

* 

**[A, E, I, O, U, A, E, I, O, U]**

* 

**Line n1 causes compilation error**

**Explanation**

Starting with JDK 7, Java allows to not specify type while initializing the ArrayList. As variable list is of List<String> type, therefore type of ArrayList is considered as String. Line n1 compiles successfully.

sublist method is declared in List interface:

List<E> subList(int fromIndex, int toIndex)

fromIndex is inclusive and toIndex is exclusive

It returns a view of the portion of this list between the specified fromIndex and toIndex. The returned list is backed by this list, so non-structural changes in the returned list are reflected in this list and vice-versa.

If returned list (or view) is structurally modified, then modification are reflected in this list as well but if this list is structurally modified, then the semantics of the list returned by this method become undefined.

If fromIndex == toIndex, then returned list is empty.

If fromIndex < 0 OR toIndex > size of the list OR fromIndex > toIndex, then IndexOutOfBoundsException is thrown.

At Line n2, list.subList(0, 4) --> [A, E, I, O] (toIndex is Exclusive, therefore start index is 0 and end index is 3].

list.addAll(list.subList(0, 4)); is almost equal to list.addAll(5, [A, E, I, O]); => Inserts A at index 5, E takes index 6, I takes index 7 and O is placed at index 8. list --> [A, E, I, O, U, A, E, I, O]

Last statement inside main(String []) method prints [A, E, I, O, U, A, E, I, O] on to the console.

Bottom of Form

Top of Form

Question 43: **Correct**

Given:

1. package com.udayankhattry.oca;
3. class Super {
4. final int NUM = -1; //Line n1
5. }
7. class Sub extends Super {
8. /\*INSERT\*/
9. }
11. public class Test {
12. public static void main(String[] args) {
13. Sub obj = new Sub();
14. obj.NUM = 200; //Line n2
15. System.out.println(obj.NUM); //Line n3
16. }
17. }

Above code causes compilation error, which modifications, done independently, enable the code to compile and on execution print 200 on to the console?

Select 4 options.

* 

**Replace /\*INSERT\*/ with byte NUM;**

* 

**Replace /\*INSERT\*/ with int NUM;**

**(Correct)**

* 

**Remove final modifier from Line n1**

**(Correct)**

* 

**Replace /\*INSERT\*/ with Object NUM;**

**(Correct)**

* 

**Replace /\*INSERT\*/ with boolean NUM;**

* 

**Replace /\*INSERT\*/ with float NUM;**

* 

**Replace /\*INSERT\*/ with short NUM;**

**(Correct)**

* 

**Replace /\*INSERT\*/ with double NUM;**

**Explanation**

Variable NUM is declared in Super class and class Sub extends Super, hence NUM can be accessed by using obj.NUM.

But as NUM Is final, hence it cannot be reassigned, therefore Line n2 causes compilation error. Let's check all the options one by one:

Remove final modifier from Line n1 => ✓ Valid option and in this case output is 200.

Replace /\*INSERT\*/ with byte NUM; => ✗ In this case, class Sub hides the variable NUM of Super class but Line n2 will still not compile as byte range is from -128 to 127 and 200 is out of range value.

Replace /\*INSERT\*/ with short NUM; => ✓ In this case, class Sub hides the variable NUM of Super class and 200 can be easily assigned to short type. In this case output is 200.

Replace /\*INSERT\*/ with int NUM; => In this case, class Sub hides the variable NUM of Super class and 200 can be easily assigned to int type. In this case output is 200.

Replace /\*INSERT\*/ with float NUM; => ✗ In this case, class Sub hides the variable NUM of Super class and 200 can be easily assigned to float type. But output in this case will be 200.0 and not 200.

Replace /\*INSERT\*/ with double NUM; => ✗ In this case, class Sub hides the variable NUM of Super class and 200 can be easily assigned to double type. But output in this case will be 200.0 and not 200.

Replace /\*INSERT\*/ with boolean NUM; => ✗ In this case, class Sub hides the variable NUM of Super class but Line n2 will still not compile as boolean type in java allows 2 values true and false. 200 is not compatible with boolean type.

Replace /\*INSERT\*/ with Object NUM; => ✓ In this case, class Sub hides the variable NUM of Super class and at Line n2, value 200 is boxed to Integer, which is then assigned to obj.NUM. So, obj.NUM refers to an instance of Integer class. Line n3 invokes toString() method of Integer class and hence 200 is printed on to the console.

Bottom of Form

Top of Form

Question 44: **Correct**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. public class Test {
4. public static void main(String[] args) {
5. int [] arr = {10, 20, 30}; //Line n1
6. int i = 0;
7. arr[i++] = arr[++i] = 40; //Line n2
8. for(int x : arr)
9. System.out.println(x);
10. }
11. }

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

* 

**40**

**40**

**30**

* 

**An exception is thrown by Line n2**

* 

**40**

**20**

**40**

**(Correct)**

* 

**10**

**40**

**40**

* 

**Compilation error at Line n2**

* 

**10**

**40**

**30**

* 

**10**

**20**

**30**

**Explanation**

At Line n1, an int [] object of three elements is created and 'arr' refers to this array object.

arr[0] = 10, arr[1] = 20 and arr[2] = 30;

Given expression at Line n2:

arr[i++] = arr[++i] = 40;

Multiple assignment operators are available, so lets group it first.

=> arr[i++] = (arr[++i] = 40); //Assignment operator is right to left associative

Above expression is valid, hence Line n2 compiles successfully.

Let's solve the expression now. Left operand is 'arr[i++]' and right operand is '(arr[++i] = 40)'. Left operand is evaluated first.

=> arr[0] = (arr[++i] = 40); //i = 1

Right hand operand is evaluated next.

=> arr[0] = (arr[2] = 40); //i = 2

=> arr[0] = 40; //i = 2, arr[2] = 40.

Hence after Line n2, arr refers to int [] object {40, 20, 40}.

Given loop prints below on to the console:

40

20

40

Bottom of Form

Top of Form

Question 45: **Incorrect**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. public class Test {
4. public static void main(String[] args) {
5. outer: for(int i = 0; i < 3; System.out.print(i)) {
6. i++;
7. inner: for(int j = 0; j < 3; System.out.print(j)) {
8. if(i > ++j) {
9. break outer;
10. }
11. }
12. }
13. }
14. }

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

* 

**Program terminates successfully but nothing is printed on to the console**

* 

**Program terminates successfully after printing 0120 on to the console**

* 

**Program terminates successfully after printing 121 on to the console**

* 

**Program terminates successfully after printing 1231 on to the console**

**(Correct)**

* 

**Program terminates successfully after printing 123 on to the console**

* 

**Compilation error**

* 

**Program terminates successfully after printing 1 on to the console**

**(Incorrect)**

* 

**Program terminates successfully after printing 12 on to the console**

**Explanation**

Basic/Regular for loop has following form:

for ( [ForInit] ; [Expression] ; [ForUpdate] ) {...}

[ForInit] can be local variable initialization or the following expressions:

Assignment

PreIncrementExpression

PreDecrementExpression

PostIncrementExpression

PostDecrementExpression

MethodInvocation

ClassInstanceCreationExpression

[ForUpdate] can be following expressions:

Assignment

PreIncrementExpression

PreDecrementExpression

PostIncrementExpression

PostDecrementExpression

MethodInvocation

ClassInstanceCreationExpression

The [Expression] must have type boolean or Boolean, or a compile-time error occurs. If [Expression] is left blank, it evaluates to true.

All the expressions can be left blank; for(;;) is a valid for loop and it is an infinite loop as [Expression] is blank and evaluates to true.

In the given code, for both the loops, `System.out.print(...)` is used as [ForUpdate] expression, which is a MethodInvocation expression and hence a valid statement.

Given code compiles successfully.

Let's check the iterations:

1st iteration of outer: i = 0. i < 3 evaluates to true.

    i = 1.

    1st iteration of inner: j = 0. j < 3 evaluates to true as j = 0. Boolean expression `i > ++j` = `1 > 1` evaluates to false. j = 1.

    2nd iteration of inner: `System.out.print(j)` prints 1 to the console. j < 3 evaluates to true as j = 1. Boolean expression `i > ++j` = `1 > 2` evaluates to false. j = 2.

    3rd iteration of inner: `System.out.print(j)` prints 2 to the console. j < 3 evaluates to true as j = 2. Boolean expression `i > ++j` = `1 > 3` evaluates to false. j = 3.

    4th iteration of inner: `System.out.print(j)` prints 3 to the console. j < 3 evaluates to false as j = 3. Control goes out of inner for loop and to the [ForUpdate] expression of outer loop.

2nd iteration of outer: `System.out.print(i)` prints 1 to the console. i < 3 evaluates to true as i = 1.

    i = 2.

    1st iteration of inner: j = 0. j < 3 evaluates to true as j = 0. Boolean expression `i > ++j` = `2 > 1` evaluates to true. j = 1. ` break outer;` takes the control out of the outer for loop.

Program terminates successfully after printing 1231 on to the console.

Bottom of Form

Top of Form

Question 46: **Correct**

Consider below codes of 4 java files:

1. //I1.java
2. package com.udayankhattry.oca;
4. public interface I1 {
5. int i = 10;
6. }
7. //I2.java
8. package com.udayankhattry.oca;
10. public interface I2 {
11. int i = 20;
12. }
13. //I3.java
14. package com.udayankhattry.oca;
16. public interface I3 extends I1, I2 { //Line n1
18. }
19. //Test.java
20. package com.udayankhattry.oca;
22. public class Test {
23. public static void main(String[] args) {
24. System.out.println(I1.i); //Line n2
25. System.out.println(I2.i); //Line n3
26. System.out.println(I3.i); //Line n4
27. }
28. }

Which of the following statements is correct?

* 

**Line n2 causes compilation error**

* 

**Line n4 causes compilation error**

**(Correct)**

* 

**Line n1 causes compilation error**

* 

**Line n3 causes compilation error**

* 

**There is no compilation error**

**Explanation**

Variable 'i' declared inside interface I1 is implicitly public, static and final and similarly variable i declared inside interface I2 is implicitly public, static and final as well.

In Java a class can extend from only one class but an interface can extend from multiple interfaces. static variables are not inherited and hence there is no issue with Line n1.

I1.i points to variable 'i' of interface I1.

I2.i points to variable 'i' of interface I2.

I3.i is an ambiguous call as compiler is not sure whether to point to I1.i or I2.i and therefore, Line n4 causes compilation error.

Bottom of Form

Top of Form

Question 47: **Correct**

Given code of Test.java file:

1. package com.udayankhattry.oca;
3. class Base {
4. String msg = "INHALE"; //Line n1
5. }
7. class Derived extends Base {
8. Object msg = "EXHALE"; //Line n2
9. }
11. public class Test {
12. public static void main(String[] args) {
13. Base obj1 = new Base(); //Line n3
14. Base obj2 = new Derived(); //Line n4
15. Derived obj3 = (Derived) obj2; //Line n5
16. String text = obj1.msg + "-" + obj2.msg + "-" + obj3.msg; //Line n6
17. System.out.println(text); //Line n7
18. }
19. }

What will be the result of compiling and executing above code?

* 

**It executes successfully and prints INHALE-INHALE-EXHALE**

**(Correct)**

* 

**Line n5 throws Exception at runtime**

* 

**Line n2 causes compilation error**

* 

**It executes successfully and prints INHALE-EXHALE-EXHALE**

* 

**None of the other options**

* 

**It executes successfully and prints INHALE-INHALE-INHALE**

* 

**Line n6 causes compilation error**

**Explanation**

Subclass overrides the methods of superclass but it hides the variables of superclass.

Line n2 hides the variable created at Line n1, there is no rules related to hiding (type and access modifier can be changed).

At Line n3, obj1 is of Base type and refers to an instance of Base class.

At Line n4, obj2 is of Base type and refers to an instance of Derived class.

At Line n5, as obj2 refers to an instance of Derived class, hence typecasting it to Derived type doesn't cause any Exception. obj3 is of Derived type and refers to an instance of Derived class.

Let's check the expression of Line n6:

obj1.msg + "-" + obj2.msg + "-" + obj3.msg;

=> (obj1.msg + "-") + obj2.msg + "-" + obj3.msg; //+ operator is left to right associative and behaves as concatenation operator as one of the operand is of String type.

=> ((obj1.msg + "-") + obj2.msg) + "-" + obj3.msg;

=> (((obj1.msg + "-") + obj2.msg) + "-") + obj3.msg;

Let's solve the expression now:

=> ((("INHALE" + "-") + obj2.msg) + "-") + obj3.msg; //obj1 is of Base type, hence obj1.msg refers to "INHALE"

=> (("INHALE-" + obj2.msg) + "-") + obj3.msg;

=> (("INHALE-" + "INHALE") + "-") + obj3.msg; //obj2 is of Base type, hence obj2.msg refers to "INHALE"

=> ("INHALE-INHALE" + "-") + obj3.msg;

=> "INHALE-INHALE-" + obj3.msg;

In above expression, left operand is of String type and right operand is of Object type, so toString() method is invoked. So, given expression is evaluated as:

=> "INHALE-INHALE-" + obj3.msg.toString();

=> "INHALE-INHALE-" + "EXHALE"; //As obj3.msg is of Object type and refers to an instance of String type, hence toString() method on "EXHALE" instance is invoked and this returns "EXHALE".

=> "INHALE-INHALE-EXHALE";

Line n7 prints INHALE-INHALE-EXHALE on to the console.

Bottom of Form

Top of Form

Question 48: **Correct**

Consider below code snippet:

1. package com.udayankhattry.oca;
3. public class Test {
4. String testNo;
5. String desc;
6. /\*
7. Other codes...
8. \*/
9. }

Which of the options are correct so that instance variables 'testNo' and 'desc' are accessible only within 'com.udayankhattry.oca' package?

* 

**Change the instance variable declarations to:**

**private String testNo;**

**private String desc;**

* 

**Change the instance variable declarations to:**

**protected String testNo;**

**protected String desc;**

* 

**Change the instance variable declarations to:**

**public String testNo;**

**public String desc;**

* 

**No changes are necessary**

**(Correct)**

**Explanation**

As member variables 'testNo' and 'desc' are declared with no explicit access specifier, this means these variables have package scope, hence these variables are accessible only to classes within the same package. Hence, no changes are necessary.

If you use private, then instance variables will not be accessible to any other classes, even within the same package.

If you use protected, then instance variables will be accessible to the subclasses outside 'com.udayankhattry.oca' package.

If you use public, then instance variables will be accessible to all the classes.

Bottom of Form

Top of Form

Question 49: **Correct**

Given code of Test.java file:

1. package com.udayankhattry.oca;
3. import java.sql.SQLException;
5. public class Test {
6. private static void getData() throws SQLException {
7. try {
8. throw new SQLException();
9. } catch (Exception e) {
10. e = new SQLException();
11. throw e;
12. }
13. }
15. public static void main(String[] args) {
16. try {
17. getData();
18. } catch(SQLException e) {
19. System.out.println("SQL");
20. }
21. }
22. }

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

* 

**SQL is printed on to the console and program terminates successfully**

* 

**Program ends abruptly**

* 

**Method main(String []) causes compilation error**

* 

**Method getData() causes compilation error**

**(Correct)**

**Explanation**

If you don't initialize variable e inside catch block using `e = new SQLException();` and simply throw e, then code would compile successfully as compiler is certain that 'e' would refer to an instance of SQLException only.

But the moment compiler finds `e = new SQLException();`, `throw e;` causes compilation error as at runtime 'e' may refer to any Exception type.

Bottom of Form

Top of Form

Question 50: **Correct**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. public class Test {
4. public static void main(String[] args) {
5. boolean b1 = 0;
6. boolean b2 = 1;
7. System.out.println(b1 + b2);
8. }
9. }

What is the result of compiling and executing Test class?

* 

**Compilation error**

**(Correct)**

* 

**true**

* 

**false**

* 

**0**

* 

**1**

**Explanation**

In Java language, boolean type can store only two values: true and false and these values are not compatible with int type.

Also + operator is not defined for boolean types. Hence, all the 3 statements inside main method causes compilation error.

Bottom of Form

Top of Form

Question 51: **Incorrect**

Given code of Test.java file:

1. package com.udayankhattry.oca;
3. interface Blogger {
4. default void blog() throws Exception {
5. System.out.println("GENERIC");
6. }
7. }
9. class TravelBlogger implements Blogger {
10. public void blog() {
11. System.out.println("TRAVEL");
12. }
13. }
15. public class Test {
16. public static void main(String[] args) {
17. Blogger blogger = new TravelBlogger(); //Line n1
18. ((TravelBlogger)blogger).blog(); //Line n2
19. }
20. }

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

* 

**An exception is thrown at runtime**

* 

**GENERIC is printed on to the console and program terminates successfully**

**(Incorrect)**

* 

**Compilation error in TravelBlogger class**

* 

**TRAVEL is printed on to the console and program terminates successfully**

**(Correct)**

* 

**Compilation error in Test class**

**Explanation**

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.

5. Cannot declare to throw unrelated checked exception.

6. May declare to throw any RuntimeException or Error.

default methods were added in Java 8 and TravelBlogger class correctly overrides the default method blog() of Blogger interface. Blogger interface compiles successfully.

At Line n1, 'blogger' is of Blogger type (supertype) and it refers to an instance of TravelBlogger class (subtype), this is polymorphism and allowed in Java. Line n1 compiles successfully.

At Line n2, blog() method is being invoked on typecasting 'blogger' to TravelBlogger and as TravelBlogger class doesn't declare to throw any checked exception, hence Line n2 compiles successfully.

As instance is of TravelBlogger type, therefore on execution, Line n2 invokes blog() method of TravelBlogger instance, which prints TRAVEL on to the console.

Bottom of Form

Top of Form

Question 52: **Correct**

Given code of Test.java file:

1. package com.udayankhattry.oca;
3. import java.io.IOException;
5. class Super {
6. Super() throws RuntimeException {
7. System.out.print("CARPE ");
8. }
9. }
11. class Sub extends Super {
12. Sub() throws IOException {
13. System.out.print("DIEM ");
14. }
15. }
17. public class Test {
18. public static void main(String[] args) throws Exception {
19. new Sub();
20. }
21. }

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

* 

**Test class executes successfully and prints DIEM CARPE on to the console**

* 

**Compilation error only in Super class**

* 

**Compilation error only in Sub class**

* 

**Compilation error in both Super and Sub classes**

* 

**Test class executes successfully and prints CARPE DIEM on to the console**

**(Correct)**

**Explanation**

It is legal for the constructors to have throws clause.

Constructors are not inherited by the Sub class so there is no method overriding rules related to the constructors but as one constructor invokes other constructors implicitly or explicitly by using this(...) or super(...), hence exception handling becomes interesting.

Java compiler adds super(); as the first statement inside Sub class's constructor:

Sub() throws IOException {

    super(); //added by the compiler

    System.out.println("DIEM");

}

super(); invokes the constructor of Super class (which declares to throw RuntimeException), as RuntimeException is unchecked exception, therefore no handling is necessary in the constructor of Sub class.

Sub class's constructor declares to throw IOException but main(String []) method handles it.

There is no compilation error and output is: CARPE DIEM

Bottom of Form

Top of Form

Question 53: **Correct**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. public class Test {
4. public static void main(String[] args) {
5. int score = 30; // Line n1
6. char grade = 'F'; // Line n2
7. if (50 <= score < 60) // Line n3
8. grade = 'D';
9. else if (60 <= score < 70) // Line n4
10. grade = 'C';
11. else if (70 <= score < 80) // Line n5
12. grade = 'B';
13. else if (score >= 80)
14. grade = 'A';
15. System.out.println(grade);
16. }
17. }

What is the result of compiling and executing Test class?

* 

**D**

* 

**F**

* 

**Compilation error**

**(Correct)**

* 

**B**

* 

**A**

* 

**C**

**Explanation**

Line n1 and Line n2 compile successfully.

Let's check the boolean expression of Line n3:

50 <= score < 60

As multiple operators are available, so let's group the operators first on the basis of precedence and associativity.

Relational operators (<, >, <= and >=) are at same level and left to right associative, hence given expression can be grouped as:

(50 <= score) < 60

< is a binary operator with two operands: (50 <= score) on the left is of boolean type and 60 on the right is of int type. But < operator is not defined for boolean, int type and hence Line n3 causes compilation error. Line n4 and Line n5 cause compilation error for the same reason.

Bottom of Form

Top of Form

Question 54: **Correct**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. public class Test {
4. public static void main(String[] args) {
5. String [] arr = {"1st", "2nd", "3rd", "4th", "5th"};
6. String place = "faraway";
7. System.out.println(arr[place.indexOf("a", 3)]); //Line n1
8. }
9. }

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

* 

**4th**

**(Correct)**

* 

**An exception is raised by Line n1**

* 

**2nd**

* 

**1st**

* 

**5th**

* 

**3rd**

**Explanation**

`int indexOf​(String str, int fromIndex)` method of String class returns the index within this string of the first occurrence of the specified substring, starting at the specified index. e.g.

"alaska".indexOf("a", 1) returns 2

"alaska".indexOf("a", 2) returns 2

"alaska".indexOf("a", 3) returns 5

In the given question, 'arr' refers to a String array of size 5. Element at index 0 refers to "1st", element at index 1 refers to "2nd" and so on.

Let's solve the given expression of Line n1:

arr[place.indexOf("a", 3)]

= arr["faraway".indexOf("a", 3)] //Starts looking for "a" from index 3 of the given String "faraway" and "a" is found at index 3.

= arr[3]

= "4th" //Array element at index 3 refers to "4th".

Hence, 4th is printed on to the console.

Bottom of Form

Top of Form

Question 55: **Correct**

Consider below codes of 2 java files:

1. //GetSetGo.java
2. package com.udayankhattry.oca;
4. public interface GetSetGo {
5. int count = 1; //Line n1
6. }
7. //Test.java
8. package com.udayankhattry.oca;
10. public class Test {
11. public static void main(String[] args) {
12. GetSetGo [] arr = new GetSetGo[5]; //Line n2
13. for(GetSetGo obj : arr) {
14. obj.count++; //Line n3
15. }
16. System.out.println(GetSetGo.count); //Line n4
17. }
18. }

Which of the following statements is correct?

* 

**Test class compiles successfully and on execution prints 5 on to the console**

* 

**Line n2 causes compilation error**

* 

**Line n4 causes compilation error**

* 

**Line n1 causes compilation error**

* 

**Line n3 causes compilation error**

**(Correct)**

* 

**Test class compiles successfully and on execution prints 6 on to the console**

**Explanation**

Variable 'count' declared inside interface GetSetGo is implicitly public, static and final. Line n1 compiles successfully.

Line n2 creates one dimensional array of 5 elements of GetSetGo type and all 5 elements are initialized to null. Line n2 compiles successfully.

Though correct way to refer static variable is by using the type name, such as GetSetGo.count but it can also be invoked by using GetSetGo reference variable. Hence, obj.count at Line n3 correctly points to the count variable at Line n1. But as variable 'count' is implicitly final, therefore obj.count++ causes compilation error. Line n3 fails to compile.

Line n4 compiles successfully as variable 'count' is implicitly static and GetSetGo.count is the correct syntax to point to 'count' variable of interface GetSetGo.

Bottom of Form

Top of Form

Question 56: **Correct**

Given code of Test.java file:

1. package com.udayankhattry.oca;
3. public class Test {
4. public static void main(String[] args) {
5. try { //outer
6. try { //inner
7. System.out.println(1/0);
8. } catch(ArithmeticException e) {
9. System.out.println("INNER");
10. } finally {
11. System.out.println("FINALLY 1");
12. }
13. } catch(ArithmeticException e) {
14. System.out.println("OUTER");
15. } finally {
16. System.out.println("FINALLY 2");
17. }
18. }
19. }

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

* 

**INNER**

**FINALLY 2**

* 

**INNER**

**FINALLY 1**

* 

**INNER**

**FINALLY 1**

**FINALLY 2**

**(Correct)**

* 

**OUTER**

**FINALLY 2**

**Explanation**

`System.out.println(1/0);` throws ArithmeticException, handler is available in inner catch-block, it executes and prints "INNER" to the console.

Once an exception is handled, no other catch block will get executed unless the exception is re-thrown.

Inner finally-block gets executed and prints "FINALLY 1" to the console.

Rule is finally-block always gets executed, so outer finally-block gets executed and prints "FINALLY 2" to the console.

Bottom of Form

Top of Form

Question 57: **Correct**

Given code of Test.java file:

1. package com.udayankhattry.oca;
3. import java.util.ArrayList;
4. import java.util.List;
5. import java.util.function.Predicate;
7. public class Test {
8. public static void main(String[] args) {
9. List<String> words = new ArrayList<>();
10. words.add("A");
11. words.add("an");
12. words.add("the");
13. words.add("when");
14. words.add("what");
15. words.add("Where");
16. words.add("whether");
18. processStringArray(words, /\*INSERT\*/);
19. }
21. private static void processStringArray(List<String> list, Predicate<String> predicate) {
22. for(String str : list) {
23. if(predicate.test(str)) {
24. System.out.println(str);
25. }
26. }
27. }
28. }

Which of the following options can replace /\*INSERT\*/ such that on executing Test class all the list elements are displayed in the output?

Select 4 options.

* 

**p -> true**

**(Correct)**

* 

**p -> p.length() < 7**

* 

**String p -> p.length() > 0**

* 

**p -> p.length() >= 1**

**(Correct)**

* 

**(String p) -> p.length() < 100**

**(Correct)**

* 

**p -> !!!!true**

**(Correct)**

* 

**p -> !!false**

**Explanation**

Interface java.util.function.Predicate<T> declares below non-overriding abstract method:

boolean test(T t);

Let's check all the options one by one:

p -> true ✓ Means test method returns true for the passed String. It will print all the elements of the List.

p -> !!!!true ✓ !!!!true => !!!false => !!true => !false => true, means test method returns true for the passed String. It will print all the elements of the List.

p -> !!false ✗ !!false => !true => false, means test method returns false for the passed String. It will not print even a single element of the list.

p -> p.length() >= 1 ✓ Means test method returns true if passed String's length is greater than or equal to 1 and this is true for all the list elements.

p -> p.length() < 7 ✗ Means test method returns true if passed String's length is less than 7 and this is not true for "whether". "whether" will not be displayed in the output.

(String p) -> p.length() < 100 ✓ Means test method returns true if passed String's length is less than 100 and this is true for all the list elements.

String p -> p.length() > 0 ✗ Round brackets or parenthesis are missing around 'String p'. This causes compilation error.

Bottom of Form

Top of Form

Question 58: **Correct**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. public class Test {
4. public static void main(String[] args) {
5. int elements = 0;
6. Object [] arr = {"A", "E", "I", new Object(), "O", "U"}; //Line n1
7. for(Object obj : arr) { //Line n2
8. if(obj instanceof String) {
9. continue;
10. } else {
11. break;
12. }
13. elements++; //Line n3
14. }
15. System.out.println(elements); //Line n4
16. }
17. }

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

* 

**0**

* 

**Compilation error at Line n3**

**(Correct)**

* 

**Compilation error at Line n1**

* 

**5**

* 

**1**

* 

**6**

* 

**3**

* 

**Compilation error at Line n2**

* 

**Compilation error at Line n4**

**Explanation**

Line n1 and Line n2 don't cause any compilation error.

if-else block uses break; and continue; statements. break; will exit the loop and will take the control to Line n4 on the other hand continue; will take the control to Line n2. In both the cases Line n3 will never be executed.

As Compiler knows about it, hence it tags Line n3 as unreachable, which causes compilation error.

Bottom of Form

Top of Form

Question 59: **Correct**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. class Currency {
4. String notation = "-"; //Line n1
6. String getNotation() { //Line n2
7. return notation;
8. }
9. }
11. class USDollar extends Currency {
12. String notation = "$"; //Line n3
14. String getNotation() { //Line n4
15. return notation;
16. }
17. }
19. class Euro extends Currency {
20. protected String notation = "€"; //Line n5
22. protected String getNotation() { //Line n6
23. return notation;
24. }
25. }
27. public class Test {
28. public static void main(String[] args) {
29. Currency c1 = new USDollar();
30. System.out.println(c1.notation + ":" + c1.getNotation());
32. Currency c2 = new Euro();
33. System.out.println(c2.notation + ":" + c2.getNotation());
34. }
35. }

What will be the result of compiling and executing above code?

* 

**$:$**

**€:€**

* 

**Compilation error in USDollar class**

* 

**-:$**

**-:€**

**(Correct)**

* 

**Compilation error in Euro class**

* 

**-:-**

**-:-**

**Explanation**

Subclass overrides the methods of superclass but it hides the variables of superclass.

Line n3 hides the variable created at Line n1 and Line n4 overrides the getNotation() method of Line n2. There is no compilation error for USDollar class as it correctly overrides getNotation() method.

Similarly, Line n5 hides the variable created at Line n1 and Line n6 overrides the getNotation() method of Line n2. There is no compilation error for Euro class as it correctly overrides getNotation() method as well.

'c1' is of Currency type, hence c1.notation refers to "-" and c1.getNotation() invokes overriding method of USDollar class and it returns "$".

Similarly, c2.notation refers to "-" and c2.getNotation() invokes overriding method of Euro class and it returns "€".

Bottom of Form

Top of Form

Question 60: **Correct**

Consider below code of TestMessage.java file:

1. package com.udayankhattry.oca;
3. class Message {
4. String msg = "LET IT GO!";
6. public void print() {
7. System.out.println(msg);
8. }
9. }
11. public class TestMessage {
12. public static void change(Message m) { //Line n5
13. m.msg = "NEVER LOOK BACK!"; //Line n6
14. }
16. public static void main(String[] args) {
17. Message obj = new Message(); //Line n1
18. obj.print(); //Line n2
19. change(obj); //Line n3
20. obj.print(); //Line n4
21. }
22. }

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

* 

**NEVER LOOK BACK!**

**NEVER LOOK BACK!**

* 

**Compilation error**

* 

**null**

**NEVER LOOK BACK!**

* 

**null**

**null**

* 

**LET IT GO!**

**LET IT GO!**

* 

**LET IT GO!**

**NEVER LOOK BACK!**

**(Correct)**

**Explanation**

Message class doesn't specify any constructor, hence Java compiler adds below default constructor:

Message() {super();}

Line n1 creates an instance of Message class and initializes instance variable 'msg' to "LET IT GO!". Variable 'obj' refers to this instance.

Line n2 prints LET IT GO! on to the console.

Line n3 invokes change(Message) method, as it is a static method defined in TestMessage class, hence `change(obj);` is the correct syntax to invoke it. Line n3 compiles successfully. On invocation parameter variable 'm' copies the content of variable 'obj' (which stores the address to Message instance created at Line n1). 'm' also refers to the same instance referred by 'obj'.

Line n6, assigns "NEVER LOOK BACK!" to the 'msg' variable of the instance referred by 'm'. As 'obj' and 'm' refer to the same instance, hence obj.msg also refers to "NEVER LOOK BACK!". change(Message) method finishes its execution and control goes back to main(String[]) method.

Line n4 is executed next, print() method is invoked on the 'obj' reference and as obj.msg refers to "NEVER LOOK BACK!", so this statement prints NEVER LOOK BACK! on to the console.

Hence in the output, you get:

LET IT GO!

NEVER LOOK BACK!

Bottom of Form

Top of Form

Question 61: **Correct**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. public class Test {
4. public static void main(String[] args) {
5. String str = "Game on"; //Line n1
6. StringBuilder sb = new StringBuilder(str); //Line n2
8. System.out.println(str.contentEquals(sb)); //Line n3
9. System.out.println(sb.contentEquals(str)); //Line n4
10. System.out.println(sb.equals(str)); //Line n5
11. System.out.println(str.equals(sb)); //Line n6
12. }
13. }

Which of the following statements is correct?

* 

**Four statements cause compilation error**

* 

**Only one statement causes compilation error**

**(Correct)**

* 

**Two statements cause compilation error**

* 

**Three statements cause compilation error**

* 

**No compilation error**

**Explanation**

There are no issues with Line n1 and Line n2, both the statements compile successfully.

String class contains contentEquals(CharSequence) method. Please note that String, StringBuilder and StringBuffer classes implement CharSequence interface, hence contentEquals(CharSequence) method defined in String class cab be invoked with the argument of either String or StringBuilder or StringBuffer.

At Line n3, `str.contentEquals(sb)` is invoked with StringBuilder argument and hence it compiles fine. On execution it would compare the contents of String object and the passed StringBuilder object. As both the String object and StringBuilder object contains same content "Game on", hence on execution, Line n3 will print true.

contentEquals method is not available in StringBuilder class and hence Line n4 causes compilation error.

equals method declared in Object class has the declaration: `public boolean equals(Object)`. Generally, equals method is used to compare different instances of same class but if you pass any other object, there is no compilation error. Parameter type is Object so it can accept any Java object.

`str.equals(sb)` => It compiles fine, String class overrides equals(Object) method but as 'sb' is of StringBuilder type so `str.equals(sb)` would return false at runtime.

`sb.equals(str)` => It also compiles fine, StringBuilder class doesn't override equals(Object) method. So Object version is invoked which uses == operator, hence `sb.equals(str)` would return false as well at runtime.

Bottom of Form

Top of Form

Question 62: **Correct**

Given code of Test.java file:

1. package com.udayankhattry.oca;
3. public class Test {
4. static String str = "KEEP IT "; //Line n1
5. public static void main(String[] args) {
6. String str = str + "SIMPLE"; //Line n2
7. System.out.println(str);
8. }
9. }

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

* 

**SIMPLE**

* 

**KEEP IT**

* 

**KEEP IT SIMPLE**

* 

**Compilation error**

**(Correct)**

**Explanation**

At Line n2, local variable 'str' shadows the static variable 'str' created at Line n1. Hence, for the expression `str + "SIMPLE"`, Java compiler complains as local variable 'str' is not initialized.

Bottom of Form

Top of Form

Question 63: **Correct**

Given code of Test.java file:

1. package com.udayankhattry.oca;
3. import java.sql.SQLException;
5. public class Test {
6. private static void getReport() throws SQLException {
7. try {
8. throw new SQLException();
9. } catch (Exception e) {
10. throw null; //Line 10
11. }
12. }
14. public static void main(String[] args) {
15. try {
16. getReport(); //Line 16
17. } catch(SQLException e) {
18. System.out.println("REPORT ERROR");
19. }
20. }
21. }

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

* 

**REPORT ERROR is printed on to the console and program terminates successfully**

* 

**Line 10 causes compilation failure**

* 

**Line 16 causes compilation failure**

* 

**Program ends abruptly**

**(Correct)**

**Explanation**

Classes in Exception framework are normal java classes, hence null can be used wherever instances of Exception classes are used, so Line 10 compiles successfully.

No issues with Line 16 as method getReport() declares to throw SQLException and main(String []) method code correctly handles it.

Program compiles successfully but on execution, NullPointerException is thrown, stack trace is printed on to the console and program ends abruptly.

If you debug the code, you would find that internal routine for throwing null exception causes NullPointerException.

Bottom of Form

Top of Form

Question 64: **Correct**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. public class Test {
4. public static void main(String [] args) {
5. String text = "ONE ";
6. System.out.println(text.concat(text.concat("ELEVEN ")).trim());
7. }
8. }

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

* 

**ONE ELEVEN ONE ELEVEN**

* 

**ONE ELEVEN ONE**

* 

**ONE ONE ELEVEN**

**(Correct)**

* 

**ONE ELEVEN**

**Explanation**

Given statement:

System.out.println(text.concat(text.concat("ELEVEN ")).trim()); //'text' refers to "ONE "

System.out.println(text.concat("ONE ELEVEN ").trim()); //As String is immutable, hence there is no change in the String object referred by 'text', 'text' still refers to "ONE "

System.out.println(("ONE ONE ELEVEN ").trim()); //'text' still refers to "ONE "

System.out.println("ONE ONE ELEVEN"); //trim() method removes the trailing space in this case

ONE ONE ELEVEN is printed on to the console.

Bottom of Form

Top of Form

Question 65: **Correct**

Consider below code fragment:

1. import java.util.\*;
3. class A{}
4. class B extends A{}
6. abstract class Super {
7. abstract List<A> get() throws IndexOutOfBoundsException;
8. }
10. abstract class Sub extends Super {
11. /\*INSERT\*/
12. }

Which of the following options replaces /\*INSERT\*/ such that there is no compilation error?

* 

**abstract List<B> get();**

* 

**abstract List<A> get() throws ArrayIndexOutOfBoundsException;**

**(Correct)**

* 

**abstract ArrayList<B> get();**

* 

**abstract ArrayList<A> get() throws Exception;**

**Explanation**

Few things to keep in mind:

1.

There are 2 rules related to return types of overriding method:

A. If return type of overridden method is of primitive type, then overriding method should use same primitive type.

B. If return type of overridden method is of reference type, then overriding method can use same reference type or its sub-type (also known as covariant return type).

2.

In case of overriding, if overridden method declares to throw any RuntimeException or Error, overriding method may or may not throw any RuntimeException but overriding method must not throw any checked exceptions.

3.

In generics syntax, Parameterized types are not polymorphic, this means even if B is subtype of A, List<B> is not subtype of List<A>. Remember this point. So below syntaxes are NOT allowed:

List<A> list = new ArrayList<B>(); OR ArrayList<A> list = new ArrayList<B>();

Let's check all the options one by one:

abstract List<A> get() throws ArrayIndexOutOfBoundsException; => ✓ It returns the same return type 'List<A>' and it is allowed to throw any RuntimeException (ArrayIndexOutOfBoundsException is RuntimeException)

abstract List<B> get(); => ✗ List<B> is not subtype of List<A>, it is not covariant return type.

abstract ArrayList<A> get() throws Exception; => ✗ As overridden method declares to throw IndexOutOfBoundsException, which is a Runtime Exception, overriding method is not allowed to declare to throw any checked Exception. Class Exception and its subclasses are checked exceptions.

abstract ArrayList<B> get(); => ✗ ArrayList<B> is not subtype of List<A>, it is not covariant return type.

Bottom of Form

Top of Form

Question 66: **Correct**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. class Super {
4. public String num = "10"; //Line n1
5. }
7. class Sub extends Super {
8. protected int num = 20; //Line n2
9. }
11. public class Test {
12. public static void main(String[] args) {
13. Super obj = new Sub();
14. System.out.println(obj.num += 2); //Line n3
15. }
16. }

What will be the result of compiling and executing above code?

* 

**It executes successfully and prints 202 on to the console**

* 

**Compilation error at Line n3**

* 

**It executes successfully and prints 22 on to the console**

* 

**It executes successfully and prints 102 on to the console**

**(Correct)**

* 

**It executes successfully and prints 12 on to the console**

* 

**Compilation error at Line n2**

**Explanation**

Subclass overrides the methods of superclass but it hides the variables of superclass.

Line n2 hides the variable created at Line n1, there is no rules related to hiding (type and access modifier can be changed).

'obj' is of Super type, hence obj.num refers to num variable at Line n1, which is of String type.

Expression at Line n3:

obj.num += 2

=> obj.num = obj.num + 2

=> obj.num = "10" + 2

=> obj.num = "102"

obj.num refers to "102" and same is printed on to the console.

Bottom of Form

Top of Form

Question 67: **Incorrect**

Given code of Test.java file:

1. package com.udayankhattry.oca;
3. public class Test {
4. public static void main(String[] args) {
5. try {
6. play();
7. return;
8. } catch(Exception ex) {
9. System.out.println(ex.getMessage());
10. return;
11. } finally {
12. System.out.println("MATCH ABANDONED");
13. }
14. System.out.println("DONE");
15. }
17. static void play() throws Exception {
18. throw new Exception("INJURED");
19. }
20. }

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

* 

**MATCH ABANDONED**

* 

**INJURED**

**MATCH ABANDONED**

**(Incorrect)**

* 

**Compilation error**

**(Correct)**

* 

**MATCH ABANDONED**

**DONE**

* 

**INJURED**

**MATCH ABANDONED**

**DONE**

* 

**INJURED**

* 

**INJURED**

**DONE**

**Explanation**

Both try and catch blocks have return; statement, which means either of the return statements will definitely get executed. Hence, compiler tags `System.out.println("DONE");` as unreachable and this causes compilation error.

Bottom of Form

Top of Form

Question 68: **Correct**

Given code of Test.java file:

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

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

* 

**Base: log()**

* 

**Derived: log()**

**(Correct)**

* 

**Compilation error in Derived class**

* 

**Compilation error in Test class**

**Explanation**

NullPointerException extends RuntimeException. Overriding method may or may not throw any RuntimeException. Only thing to remember is that if overridden method throws any unchecked exception or Error, then overriding method must not throw any checked exceptions.

So, method log() in Derived class correctly overrides Base class's method.

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

Bottom of Form

Top of Form

Question 69: **Correct**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. public class Test {
4. int i1 = 10;
5. static int i2 = 20;
7. private void change1(int val) {
8. i1 = ++val; //Line n1
9. i2 = val++; //Line n2
10. }
12. private static void change2(int val) {
13. i1 = --val; //Line n3
14. i2 = val--; //Line n4
15. }
17. public static void main(String[] args) {
18. change1(5); //Line n5
19. change2(5); //Line n6
20. System.out.println(i1 + i2); //Line n7
21. }
22. }

Which of the following statements are correct regarding above code?

Select 3 options.

* 

**Line n7 causes compilation error**

**(Correct)**

* 

**Above code prints 30 on execution**

* 

**Line n2 causes compilation error**

* 

**Line n1 causes compilation error**

* 

**Above code prints 8 on execution**

* 

**Line n5 causes compilation error**

**(Correct)**

* 

**Line n6 causes compilation error**

* 

**Line n4 causes compilation error**

* 

**Above code compiles successfully**

* 

**Line n3 causes compilation error**

**(Correct)**

**Explanation**

i1 is an instance variable and i2 is a static variable.

Instance method can access both instance and static members. Hence, Line n1 and Line n2 compile successfully.

Static method can access only static members. Hence, Line n3 [accessing instance variable i1], Line n5 [accessing instance method change1(int)] and Line n7 [accessing instance variable i1] cause compilation error.

Bottom of Form

Top of Form

Question 70: **Incorrect**

Consider below code of Test.java file:

1. package com.udayankhattry.oca;
3. public class Test {
4. public static void main(String[] args) {
5. int var = 3;
6. String [][] arr = new String[--var][var++]; //Line n1
7. arr[1][1] = "X"; //Line n2
8. arr[1][2] = "Y"; //Line n3
9. for(String [] arr1 : arr) {
10. for(String s : arr1) {
11. if(s != null)
12. System.out.print(s);
13. }
14. }
15. }
16. }

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

* 

**It causes compilation error at single statement**

**(Incorrect)**

* 

**It prints XY on to the console and program terminates abruptly**

* 

**It causes compilation error at multiple statements**

* 

**It throws an exception at runtime**

**(Correct)**

* 

**It prints XY on to the console and program terminates successfully**

**Explanation**

Line n1:

String [][] arr = new String[--var][var++]; //var = 3

Access array element operator [] is left to right associative.

=> String [][] arr = new String[2][var++]; //var = 2, var is decremented first and then used in the expression.

=> String [][] arr = new String[2][2]; //var = 3, value of var is used first and then it is incremented by 1

Hence, arr refers to 2-dimensional String array object {{null, null}, {null, null}}.

At Line n2, arr[1][1] = "X"; assigns "X" to element at index [1][1], therefore arr --> {{null, null}, {null, "X"}}

At Line n3, arr[1][2] = "Y"; causes ArrayIndexOutOfBoundsException as 2nd index 2 is out of range.

As Line n3 throws Exception at runtime, hence for loop will not be executed.

Bottom of Form

Top of Form

Question 71: **Correct**

\_\_\_\_\_\_\_\_\_ modifier is most restrictive and \_\_\_\_\_\_\_\_\_\_ modifier is least restrictive.

Which of the following options (in below specified order) can be filled in above blank spaces?

* 

**default (with no access modifier specified), protected**

* 

**private, public**

**(Correct)**

* 

**default (with no access modifier specified), public**

* 

**protected, public**

* 

**public, private**

**Explanation**

'private' is most restrictive, then comes 'default (with no access modifier specified)', after that 'protected' and finally 'public' is least restrictive.

Bottom of Form

Top of Form

Question 72: **Incorrect**

Given code of Test.java file:

1. package com.udayankhattry.oca;
3. public class Test {
4. private static void test() throws Exception {
5. throw new Exception();
6. }
8. public static void main(String [] args) {
9. try {
10. test();
11. } finally {
12. System.out.println("GAME ON");
13. }
14. }
15. }

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

* 

**Compilation error**

**(Correct)**

* 

**GAME ON is printed to the console, stack trace is printed and then program ends abruptly**

**(Incorrect)**

* 

**GAME ON is printed to the console and program ends normally**

* 

**GAME ON is printed to the console, stack trace is printed and then program ends normally**

**Explanation**

Method test() throws Exception (checked) and it declares to throw it, so no issues with method test().

But main(String []) method neither provides catch handler nor throws clause and hence main(String []) method causes compilation error.

Handle or Declare rule should be followed for checked exception if you are not re-throwing it.

Bottom of Form

Retake test

**Continue**