Explanation:
Correct answer is option B.

Array is declared with 10 elements. Then the code tries to access the 11th index element of the array which throws an standard exception 'java.lang.ArrayIndexOutOfBoundsException'.

Explanation :
Option A and C are the correct answer.

In a method declaration, the keyword throws is used. So here at line 1 we have to use option A.

To actually throw an exception, the keyword throw is used and a new exception is created, so at line 2 we have to use throw and new keywords, which is option C. Finally it will look like;

```
public void method() throws Exception {

        throw new Exception();

}
```

Explanation :
Option B is the correct answer.

In the given statement we can see that we have passed a negative value for creating int array, which results in a NegativeArraySizeException. Hence, option B is correct.

Option A is incorrect as it is thrown when an application attempts to use null in a case where an object is required.

Option D is incorrect as IndexOutOfBoundsException error is thrown to indicate that an index of some sort (such as to an array, to a string, or to a vector) is out of range.

Explanation :
Option D is the correct answer.

Curly braces are optional for 'for loop' . If for loop doesn't contain curly braces , only the statement following the for loop comes within for loop body.

The scope of loop variables remains within the loop only. So, in this case, the scope of the loop variable x declared at line 5, limited to that for loop. The code is trying to access that variable in line 7, which is out of the scope of the variable x, causes a compile time error. So, compilation fails due to the error at line 7. Hence, option D is correct.

Options A and B are incorrect; since code fails to compile.

Option A is correct.

The compiler is invoked by the javac command. When compiling a java class, you must include the file name, which houses the main classes including the .java extension. So, we have to use the command in option A to run Main.java file.

To execute java program, we can use java command but can't use it for compiling.

Option E is the correct answer.

At line 5, we have created a wrapper object of the double by passing 120D, which is convertible to a Double. So, there won't be any exception. But if you check carefully, you can see that variable number is declared inside try block; so, the scope of the variable number is limited to that block only, so trying to access it outside causes a compile time error. Compilation fails due to an error at line 9.

Hence, option E is correct.

Explanation:

Correct answers are options A & B.

Option A : for each loop condition
Option B : generic for loop condition

Rest of them are generic and do not form a validated string.

Explanation :
Option B is the correct answer.

The switch use equals method of string. So, in this case passing "A" only matches with the case with "A", so only Capital A will be printed as output. Hence option B is correct.

Explanation :
Option C is the correct answer.

Here, we have to use two ternary operators altogether. So, firstly, we will check the x > 10, as follows;

```
x>10?">": (when condition true)
```

Now, we have to use another to check if x<10 as follows;

```
x<10?"<": "="
```

We can combine these two by putting last ternary statement in the false position of ternary statement as follows;

```
x>10?">": x<10?"<": "="
```

As explained above, option C is correct.

The correct answer is options B & D.

We need to select java feature which can only be implemented with multiple classes. Among the given options, inheritance and composition can be only be implemented with multiple classes. Refactoring is not java feature, It is automatically gets eliminated. Reflection concept can be used with single class alone.

Hence options B, D are correct.

Explanation :
Option B is the correct answer.

Option A is incorrect as we can't use abstract with a non-abstract method. (here method has method body.)

Option C is incorrect as when overriding method we can't use more restrictive access modifier, so trying to use private to override default access level method causes a compile time error.

Option D is incorrect as default methods (not methods with default access level) are allowed only in interfaces.

Option E is incorrect as method all ready has void as return type, so we can't add int there.

Option B is correct as we can use final there, since the method is non-abstract.

Explanation :
Option C is the correct answer.

From java SE 8, we can use static and/or default methods in interfaces, but they should be non-abstract methods. So, in this case, using default in blank is completely legal, Hence option C is correct.

Option A is incorrect as given method is not abstract, so can't use abstract there.

Options B and E are incorrect as we can't have non-abstract method interface if they are not default or static.

Option C is the correct answer.

Both name and pass variables are instance variables, and we haven't given them any values. So, they take their default values. For boolean, default value is false and for string, which is not a primitive type, default is null. So, at line 7, null will printed as the value of the variable name, and at line 8 false will be printed. Hence, Option C is correct.

As explained above options A, B and D are incorrect.

Code compiles fine so option E is incorrect.

Option B and D are the correct answer.

Since the variables i and j are floats, resultant will be float type too. So, we have to use float or primitive type which can hold float, such as double. it has a wider range and also can hold floating point numbers. Hence, we can use double or float for the blank.

As explained above options B and D are correct.

long and int can't be used with floating point numbers so option A is incorrect.

Option E is incorrect as it has a smaller range and also, can't be used with floating point numbers.

Explanation :
Option D is the correct answer.

The Integer class valueOf() returns an Integer from given string. But we need to pass a string which has correct format for integer otherwise, it will throw a NumberFormatException. In this case, we have passed a string which is not an integer value (since what we passed is a fractional number), so option D is correct.

Explanation:
Correct answer is option B.

Compilation fails and the error message is displayed regarding required and currently found number of arguments.

Explanation :
Option D is the correct answer.

Statement I is correct as the default constructor contains super() call.

Statement II is incorrect as we can use any access modifier with a constructor.

Statement III is correct as constructor can't have return-type, even void; If so, it will be a method.

So, option D is correct.

Option B is the correct answer.

Given method is declared as default method so we can declare it only inside an interface. Hence option B is correct and option D is incorrect.

Option A is incorrect as it is valid method. Option C is incorrect as return type is void, which means we can't return anything.

Option E is the correct answer.

Code fails to compile because there is no method called concat present in the StringBuilder class. The concat method lies in String class. Hence, option E is correct. Here, we should have used append method of StringBuilder class, in that case, option B would be correct.

Explanation :
Option C is the correct answer.

The LocalTime class doesn't have any visible constructor, so we can't use new keyword with them. So options A and B are incorrect.

To get the current time, we can call now method on LocalTime class. So, option C is correct.

Option D is incorrect as there is no method called today in LocalTime interface.

Explanation :
Option A is the correct answer.

Following method has been used with LocalDate class;

public static LocalDate of(int year, int month, int dayOfMonth)

Here we need to remember that month is not zero based so if you pass 1 for month, then month will be January.

Then we have used period object of 1 day and add to date object which makes current date to next day, so final output is 2015-03-27. Hence option A is correct.

Explanation :
Option A is the correct answer.

Following method has been used with LocalDate class;

public static LocalDate of(int year, int month, int dayOfMonth)

Here we need to remember that month is not zero based so if you pass 1 for month, then month will be January.

Then we have used period object of 1 day and add to date object which makes current date to next day, so final output is 2015-03-27. Hence option A is correct.

Explanation :
Option B is the correct answer.

Here we use lambda expression for concreting abstract method of Multi-interface. A lambda expression is composed of three parts.

Argument List   *Arrow Token*   *Body*

The body can be either a single expression or a statement block. In the expression form, the body is simply evaluated and returned. In the block form, the body is evaluated like a method body and a return statement returns control to the caller of the anonymous method. But remember return statement is NOT an expression so, in a lambda expression, you MUST enclose statements in braces ({...}). Also, you can omit the data type of the parameters in a lambda expression.

As explained above, Option B is correct.

Option A is incorrect as we have missed the semicolon at the end of the statement.

Options C and D are incorrect because they are invalid lambda expressions.

Explanation :
Option C is the correct answer.

In the output, we can see that only odd numbers are present. So, we need to remove all even numbers to get the expected output. From java SE 8, there is new method call removeIf which takes predicate object and remove elements which satisfy predicate condition.

The Predicate has functional method call, take object. Check if the given condition met or not, if met it returns true, otherwise false. Option C we have passed the correct lambda expression to check whether the number is odd or even that matches to the functional method of predicate interface.

Option A is incorrect as it is an invalid lambda expression. Option B is incorrect as it removes all odd numbers.

Option D is incorrect as there is no remove method that takes a predicate as an argument.

Explanation :
Option D is the correct answer.

Code fails to compile as we can't use primitive for collections type. So, in this code, when we try to use int at line 7, causes a compile error. We should use wrapper Integer there. So option D is correct.

The option C is the correct answer.

An ArrayList can be used when the number of elements in the list is not precisely known. Once the number of elements in the list is fixed, ArrayList can be converted into an Array. And the given code describes this conversion. you may want to convert an ArrayList to an array & the mentioned code gives an example of how these are converted.

public T[] toArray(T[] a)
Returns an array containing all of the elements in this list in proper sequence (from first to last element); the runtime type of the returned arrayis that of the specified array. If the list fits in the specified array, it is returned therein. Otherwise, a new array is allocated with the runtime type of the specified array and the size of this list.
Below are the examples to understand functionality of "toArray()" method.

case 1: If the list fits in the specified array, it is returned therein.

```
import java.util.ArrayList;
import java.util.Arrays;
public class Test {
 public final static void main(String[] args) {
  ArrayList whizlArray = new ArrayList<>();
  whizlArray.add("coke");
  whizlArray.add("pepsi");
  whizlArray.add("miranda");
  String[] ws1 = new String[whizlArray.size()];
  String[] ws2 = whizlArray.toArray(ws1);
  System.out.println("ws1 == ws2:" + (ws1 == ws2));
  System.out.println("ws1:" + Arrays.toString(ws1));
  System.out.println("ws2:" + Arrays.toString(ws2));
 }
}
```

In the above code ArrayList has 3 elements , "ws1" array size is also 3. "toArray()" method doesn't create new Array. "toArray()" method initializes all ArrayList elements into "ws1" array. we also assigning to "ws2". It also refers to same array.

Output

```
ws1 == ws2:true
ws1:[coke, pepsi, miranda]
ws2:[coke, pepsi, miranda]
```

case 2: Otherwise, a new array is allocated with the runtime type of the specified array and the size of this list.

```
import java.util.ArrayList;
import java.util.Arrays;
public class Test {
 public final static void main(String[] args) {
  ArrayList whizlArray = new ArrayList<>();
  whizlArray.add("coke");
  whizlArray.add("pepsi");
  whizlArray.add("miranda");
  String[] ws1 = new String[1];
  String[] ws2 = whizlArray.toArray(ws1);
  System.out.println("ws1 == ws2:" + (ws1 == ws2));
  System.out.println("ws1:" + Arrays.toString(ws1));
  System.out.println("ws2:" + Arrays.toString(ws2));
 }
}
```

In the above code ArrayList has 3 elements , but "ws1" array size is 1. So, "toArray()" method create new Array with type as "ws1" array type and assigns to "ws2".

Output:

```
ws1 == ws2:false
ws1:[null]
ws2:[coke, pepsi, miranda]
```