09 Java
Working with Java API

Problema 1

In Java, Strings are immutable. A direct implication of this is…

You had to select 2 options

- you cannot call methods like "1234".replace('1', '9'); and expect to change the original String.
    - calling such methods do not change this object. They create a new String object.
- you cannot change a String object, once it is created.
- you can change a String object only by the means of its methods.
- you cannot extend String class.
    - That's because it is final, not because it is immutable. You can have a final class whose objects are mutable.
- you cannot compare String objects.
    - String class implements Comparable interface.

Problema 2

Which of these expressions will return true?
You had to select 4 options

- "hello world".equals("hello world")
- "HELLO world".equalsIgnoreCase("hello world")
    - equalsIgnoreCase() method treats both cases (upper and lower) as same.
- "hello".concat(" world").trim().equals("hello world")
    - "hello".concat(" world") will return "hello world" and trim() won't do any change because there is no space at the beginning or end of the string.
- "hello world".compareTo("Hello world") < 0
    - Notice that the Strings differ at the first position. The value returned by compareTo is (Unicode value of the left hand side - Unicode value of the right hand side).
    - Although not required for the exam, it is good to know that for English alphabets, the unicode value of any lower case letter is always 32 more than the unicode value of the same letter in upper case. So, 'a' - 'A' or 'h' - 'H' is 32.
    - Note that int value of ASCII 'a' is 97, while that of 'A' is 65.
- "Hello world".toLowerCase( ).equals("hello world")
    - toLowerCase() converts all uppercase letters to lower case.

Explanation:
compareTo() does a lexicographical (like a dictionary) comparison. It stops at the first place where the strings have different letters.
If left hand side is bigger, it returns a positive number otherwise it returns a negative number. The value is equal to the difference of their unicode values.
If there is no difference then it returns zero. In this case, it will return ('h' - 'H') which is 32.

Problema 3

What will the following code print?

```
String abc = "";
abc.concat("abc");
abc.concat("def");
System.out.print(abc);
```

You had to select 1 option

- abc
- abcdef
- def
- It will print empty string (or in other words, nothing).
- It will not compile because there is no concat() method in String class.

Explanation

Strings are immutable so doing `abc.concat("abc")` will create a new string `"abc"` but will not affect the original string `""`.

Problema 4 - Hello, how are you?

What will the following code print when compiled and run?

```
class StringWrapper {
   private String theVal;
   public StringWrapper(String str){ this.theVal = str; }
}
public class Tester{
    public static void main(String[] args) {
        StringWrapper sw = new StringWrapper("How are you?");
        StringBuilder sb = new StringBuilder("How are you?");
        System.out.println("Hello, "+sw);
        System.out.println("Hello, "+sb);
   }
}
```

You had to select 1 option

- `Hello, How are you?`
  `Hello, How are you?`
- `Hello, StringWrapper@<hashcode>`
  `Hello, How are you?`
- `Hello, How are you?`
  `Hello, StringBuilder@<hashcode>`
- `Hello, How are you?`
  `Hello, java.lang.StringBuilder@<hashcode>`
- `Hello, StringWrapper@<hashcode>`
  `Hello, java.lang.StringBuilder@<hashcode>`

Explanation

1. When one of the operands of the + operator is a `String` and other is an object (other than `String`), `toString` method is called on the other operand and then both the Strings are concatenated to produce the result of the operation.
2. `Object` class contains an implementation of `toString` that returns the name of the class (including the package name) and the hash code of the object in the format `<classname>@<hashcode>`. For example, `System.out.println("Hello, "+new Object());` will print `Hello, java.lang.Object@3cd1a2f1`, where `3cd1a2f1` is the hash code of the object.
3. `StringBuilder` class provides its own implementation of `toString` method, which returns the `String` value of its contents.

In this question, `StringWrapper` class does not implement `toString` method and so `Object` class's version is used.

Problema 5 - fullPhoneNumber

Assuming that the following method will always be called with a phone number in the format ddd-ddd-dddd (where d stands for a digit), what can be inserted at //1 so that it will return a String containing the same number except its last four digits will be masked with xxxx?

```
public static String hidePhone(String fullPhoneNumber){
  //1 Insert code here
}
```

You had to select 3 options

- `return new StringBuilder(fullPhoneNumber).substring(0, 8)+"xxxx";`
- `return new StringBuilder(fullPhoneNumber).replace(8, 12, "xxxx").toString();`
- `return new StringBuilder(fullPhoneNumber).append("xxxx", 8, 12).toString();`
    - This will actually throw an IndexOutOfBoundsException because the call to append will look for characters starting from index 8 to 11 in string "xxxx", which has only 4 characters.
- `return new StringBuilder("xxxx").append(fullPhoneNumber, 0, 8).toString();`
    - This will return xxxxddd-ddd-.
- `return new StringBuilder("xxxx").insert(0, fullPhoneNumber, 0, 8).toString();`

Explanation

This is a simple question if you know how the various methods of StringBuilder operate. You need to go through the JavaDoc API descriptions of the methods used in this question. This is important for the exam. The following are the details for your convenience -

```
Sout(+-+----------------------------------------------------------------+-+)
```

```
public StringBuilder append(CharSequence s, int start, int end)
```

Appends a subsequence of the specified CharSequence to this sequence.
Characters of the argument s, starting at index start, are appended, in order, to the contents of this sequence up to the (exclusive) index end. The length of this sequence is increased by the value of end - start.
Let n be the length of this character sequence just prior to execution of the append method. Then the character at index k in this character sequence becomes equal to the character at index k in this sequence, if k is less than n; otherwise, it is equal to the character at index k+start-n in the argument s.
If s is null, then this method appends characters as if the s parameter was a sequence containing the four characters "null".

Parameters:
s - the sequence to append. start - the starting index of the subsequence to be appended. end - the end index of the subsequence to be appended.

Returns:
a reference to this object.

Throws:
IndexOutOfBoundsException - if start is negative, or start is greater than end or end is greater than s.length()

```
Sout(+-+------------------------------------------------------------+-+)
```

```
public StringBuilder insert(int dstOffset, CharSequence s, int start, int
end)
```

Inserts a subsequence of the specified CharSequence into this sequence.
The subsequence of the argument s specified by start and end are inserted, in order, into this sequence at the specified destination offset, moving up any characters originally above that position. The length of this sequence is increased by end - start.
The character at index k in this sequence becomes equal to:
the character at index k in this sequence, if k is less than dstOffset
the character at index k+start-dstOffset in the argument s, if k is greater than or equal to dstOffset but is less than dstOffset+end-start
the character at index k-(end-start) in this sequence, if k is greater than or equal to dstOffset+end-start
The dstOffset argument must be greater than or equal to 0, and less than or equal to the length of this sequence.
The start argument must be nonnegative, and not greater than end.
The end argument must be greater than or equal to start, and less than or equal to the length of s.
If s is null, then this method inserts characters as if the s parameter was a sequence containing the four characters "null".

Parameters:
dstOffset - the offset in this sequence. s - the sequence to be inserted. start - the starting index of the subsequence to be inserted. end - the end index of the subsequence to be inserted.

Returns:
a reference to this object.

Throws:
IndexOutOfBoundsException - if dstOffset is negative or greater

```
Sout(+-+------------------------------------------------------------+-+)
```

```
public StringBuilder replace(int start, int end, String str)
```

Replaces the characters in a substring of this sequence with characters in the specified String. The substring begins at the specified start and extends to the character at index end - 1 or to the end of the sequence if no such character exists. First the characters in the substring are removed and then the specified String is inserted at start. (This sequence will be lengthened to accommodate the specified String if necessary.)

Parameters:
start - The beginning index, inclusive. end - The ending index, exclusive. str - String that will replace previous contents.

Returns:
This object.

Throws:
StringIndexOutOfBoundsException - if start is negative, greater than length(), or greater than end.

```
Sout(+-+------------------------------------------------------------+-+)
```

```
public String substring(int start, int end)
```

Returns a new String that contains a subsequence of characters currently contained in this sequence. The substring begins at the specified start and extends to the character at index end - 1.

Parameters:
start - The beginning index, inclusive. end - The ending index, exclusive.

Returns:
The new string.

Throws:
StringIndexOutOfBoundsException - if start or end are negative or greater than length(), or start is greater than end.

```
Sout(+-+------------------------------------------------------------+-+)
```

Problema 6

In which sequence will the characters a, b, and, c be printed by the following program?

```
import java.util.* ;
public class ListTest{
   public static void main(String args[]){
      List s1 = new ArrayList( );
      s1.add("a");
      s1.add("b");
      s1.add(1, "c");
      List s2 = new ArrayList(  s1.subList(1, 1) );
      s1.addAll(s2);
      System.out.println(s1);
   }
}
```

You had to select 1 option

- The sequence a, b, c is printed.
- The sequence a, b, c, b is printed.
- The sequence a, c, b, c is printed.
- The sequence a, c, b is printed.
    - add(1, "c") will insert 'c' between 'a' and 'b' . subList(1 , 1) will return an empty list.
- None of the above.

Explanation:
First, "a" and "b" are appended to an empty list. Next, "c" is added between "a" and "b".
Then a new list s2 is created using the sublist view allowing access to elements from index 1 to index 1(exclusive) (i.e. no elements). (Note that if fromIndex and toIndex arguments t subList method are equal (as is the case in this question), subList returns an empty list.).
Now, s2 is added to s1.
So s1 remains :a, c, b

Problema 7

Which of these statements concerning the `charAt()` method of the String class are true?

You had to select 2 options

- The `charAt( )` method can take a char value as an argument.
    - Yes, it can because it takes an int and char will be implicitly promoted to int.
- The `charAt( )` method returns a Character object.
    - It returns char.
- The expression `char ch = "12345".charAt(3)` will assign 3 to ch.
    - It will assign 4 as indexing starts from 0.
- The expression `char ch = str.charAt(str.length())` where str is `"12345"`, will assign 3 to `ch`.
    - It will throw IndexOutOfBoundsException as str.length() is 5 and there is no str.charAt(5);
- The index of the first character is 0.
- It throws `StringIndexOutOfBoundsException` if passed a value higher than or equal to the length of the string (or less than 0).
- It throws `ArrayIndexOutOfBoundsException` if passed an value higher than or equal to the length of the string (or less than 0).

Explanation

Since indexing starts with 0, the maximum value that you can pass to `charAt` is length-1.

As per the API documentation for `charAt`, it throws `IndexOutOfBoundsException` if you pass an invalid value.

Both - `ArrayIndexOutOfBoundsException` and `StringIndexOutOfBoundsException`, extend `IndexOutOfBoundsException` and although in practice, the `charAt` method throws `StringIndexOutOfBoundsException`, it is not a valid option because the implementation is free to throw some other exception as long as it is an `IndexOutOfBoundsException`. There are questions in the exam on this aspect.

Problema 8

What will the following program print?

```
public class TestClass{
  static String str = "Hello World";
  public static void changeIt(String s){
    s = "Good bye world";
  }
  public static void main(String[] args){
    changeIt(str);
    System.out.println(str);
  }
}
```

You had to select 1 option

- "Hello World"
- "Good bye world"
- It will not compile.
- It will throw an exception at runtime.
- None of the above.

Explanation

Theoretically, java supports Pass by Value for everything (i.e. primitives as well as Objects).
Remember that:
1. Primitives are always passed by value.
2. Object "references" are passed by value. So it looks like the object is passed by reference but actually it is the value of the reference that is passed.

An example:

```
Object o1 = new Object();
```

Let us say, this object is stored at memory location `15000`. Since `o1` actually stores just the address of the memory location where the object is stored, it contains `15000`.

Now, when you call `someMethod(o1);` the value `15000` is passed to the method. Therefore, this is what happens inside the method `someMethod()`:

```
someMethod( Object localVar) {
```

`localVar` now contains `15000`, which means `localVar` also points to the same memory location where the object is stored. Therefore, when you call a method on `localVar`, it will be executed on the same object.

However, when you change the value of `localVar` itself, for example, if you do `localVar=null`, then `localVar` starts pointing to a different memory location. But the original variable `o1` still contains `15000` so it still points to the same object.
}

Problema 9

What will the following code print?

```
List s1 = new ArrayList( );
s1.add("a");
s1.add("b");
s1.add("c");
s1.add("a");
if(s1.remove("a")){
    if(s1.remove("a")){
        s1.remove("b");
    }else{
        s1.remove("c");
    }
}
System.out.println(s1);
```

You had to select 1 option

- [b]
- [c]
- [b, c, a]
- [a, b, c, a]
- Exception at runtime

Explanation

ArrayList's `remove(Object )` method removes the first occurence of the given element and returns true if found. It does not remove all occurences of the element. In this case, the first call to `s1.remove("a");` will remove only the first "a" and return true, the second call to `remove("a")` will remove the second "a" and return true. Finally, the call to remove("b") will remove "b". Therefore, only "c" will be left in the list.

The JavaDoc API description of this method is important for the exam -

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

Problema 10

What will the following code print when compiled and run?

```
import java.time.*;
import java.time.format.*;
public class DateTest{
  public static void main(String[] args){ //1
        LocalDateTime greatDay = LocalDateTime.parse("2015-01-01");//2
        String greatDayStr =
greatDay.format(DateTimeFormatter.ISO_DATE_TIME); //3
        System.out.println(greatDayStr);//4
   }
}
```

You had to select 1 option

- //1 will not compile because of lack of throws clause.
    - Operations in the new date/time related classes throw java.time.DateTimeException, which extends from RuntimeException.
    - Therefore, this exception is not required to be caught or declared in the throws clause.
- //2 will not compile because of invalid date string.
    - The given date string does not contain a time component and so it cannot be parsed by LocalDateTime. However, this is a run time issue and not a compile time one.
- //2 will throw an exception at run time.
    - It will throw a DateTimeException because it doesn't have time component.
    - Exception in thread "`main`" `java.time.format.DateTimeParseException`:
    - Text '`2015-01-01`' could not be parsed at index 10.
    - A String such as `2015-01-01T17:13:50` would have worked.
- It will print `2015-01-01T00:00:00`
- It will print `null`.

Problema 11

Which of the following options correctly add 1 month and 1 day to a given LocalDate -

```
public LocalDate process(LocalDate ld){
   //INSERT CODE HERE
   return ld2;
}
```

You had to select 1 option

- `LocalDate ld2 = ld.plus(Period.ofMonths(1).ofDays(1));`
  - ofXXX are static methods of Period class. Therefore, `Period.ofMonths(1).ofDays(1)` will give you a Period of only 1 day. The previous call to ofMonths(1) does return an instance of Period comprising 1 month but that instance is irrelevant because ofDays is a static method.
- `LocalDate ld2 = ld.plus(new Period(0, 1, 1));`
  - None of the new date related classes have public constructors. So using new to create their instances would be invalid.
- `LocalDate ld2 = ld.plus(new Period(31)).plus(new Period(1));`
  - None of the new date related classes have public constructors. So using new to create their instances would be invalid.
  - Further, a month is not necessarily equal to 31 days. The number of days added to a given month depends on the month to which you are adding a month.
  - For example, if you add 1 month to 1st January, you will get 1 February i.e. 31 days are added. But if you add 1 month to 1st February, you will still get 1 March i.e. only 28 days are added, (or if it is a leap year, 29 days).
- `LocalDate ld2 = ld.plus(Period.of(0, 1, 1));`
  - public `static Period of(int years, int months, int days)`
  - Obtains a Period representing a number of years, months and days.
  - This creates an instance based on years, months and days.

Problema 12

Which of these expressions will obtain the substring "456" from a string defined by String str = "01234567"?

You had to select 1 option

`str.substring(4, 7)`
`str.substring(4)`
It will return "4567".
`str.substring(3, 6)`
It will return "345".
`str.substring(4, 6)`
It will return "45".
`str.substring(4, 3)`
Exception in thread "main" `java.lang.StringIndexOutOfBoundsException`: String index out of range: -1

Explanation:
Read this carefully:
public String substring(int beginIndex, int endIndex)
Returns a new string that is a substring of this string. The substring begins at the specified beginIndex and extends to the character at index endIndex - 1. Thus the length of the substring is endIndex-beginIndex.

`"hamburger".substring(4, 8)` returns `"urge"`
`"smiles".substring(1, 5)` returns `"mile"`
`"unhappy".substring(2)` returns `"happy"`
`"Harbison".substring(3)` returns `"bison"`
`"emptiness".substring(9)` returns `""` `(an empty string)`

Problema 13 - String isa string

What will be the output of the following program (excluding the quotes)?

```
public class SubstringTest{
   public static void main(String args[]){
      String String = "string isa string";
      System.out.println(String.substring(3, 6));
   }
}
```

You had to select 1 option

- It will not compile.
    - String String = "String"; is a perfectly valid syntax! Java does not allow variables to have the same name as keywords (if, for, else, while, class etc.) and literals (true, false, and, null) but there is no restriction on naming variables after the names of classes.
    - In this case, therefore, String is a valid variable name. In case there is a conflict between variable name and class name, if the variable is in scope then it is the variable that is accessed instead of the class. For example, consider the following code
        - `String String  = "test";`
        - `System.out.println(String.length());//prints 4`
    - Here, the variable String is in scope and therefore the length of the string pointed to by the variable String is printed. The compiler does not confuse it with the class name String.
- "ing is"
- "ing isa"
- "ing " (There is a space after g)
- None of the above.
    - It will print 'ing'. (No space after 'g')

Explanation

Remember, indexing always starts from 0.

`"hamburger".substring(4, 8)` returns `"urge"`

`"smiles".substring(1, 5)` returns `"mile"`

`"01234".substring(1, 1)` returns `""`. This shows that beginIndex and endIndex can be same.

`"01234".substring(1, 0)` throws `java.lang.StringIndexOutOfBoundsException`. This shows that endIndex cannot be less than beginIndex.

Parameters:
beginIndex - the beginning index, inclusive.
endIndex - the ending index, exclusive.
Returns:
the specified substring.
Throws:
`IndexOutOfBoundsException` - if the beginIndex is negative, or endIndex is larger than the length of this String object, or beginIndex is larger than endIndex.

Problema 14

What will the following code print when compiled and run?

```
import java.util.*;
public class TestClass {
    public static void main(String[] args) throws Exception {
        ArrayList<String> al = new ArrayList<String>();
        al.add("111");
        al.add("222");
        System.out.println(al.get(al.size()));
     }
}
```

You had to select 1 option

- It will not compile.
- It will throw a NullPointerException at run time.
- It will throw an `IndexOutOfBoundsException` at run time.
    - size() method of ArrayList returns the number of elements. Here, it returns 2. Since numbering in ArrayList starts with 0. al.get(2) will cause an IndexOutOfBoundsException to be thrown because only 0 and 1 are valid indexes for a list of size 2.
- 222
- null

Problema 15 - “hello world”

Which of these are valid expressions to create a string of value "hello world" ?
You had to select 3 options

- " hello world".trim()
    - trim() removes starting and ending spaces.
- ("hello" + new String("world"))
    - It will create helloworld. No space between hello and world.
- "hello".concat(" world")
- new StringBuilder("world").insert(0, "hello ").toString();
- new StringBuilder("world").append(0, "hello ").toString();
    - 1. append adds the argument to the end.
    - 2. It doesn't take an int as its first argument.
- new StringBuilder("world").append("hello ", 0 , 6).toString();
    - There is an append method that takes two ints as shown here but the int parameters are to determine the portion of the String that is to be appended to the target.
    - That portion will still be appended to the end of the target.
- new StringBuilder("world").add(0, "hello ").toString();
    - There is no add method in StringBuilder.

Problema 16 - setLenght()

What will be the result of attempting to compile and run the following program?

```
public class TestClass{
   public static void main(String args[ ] ){
      StringBuilder sb = new StringBuilder("12345678");
      sb.setLength(5);
      sb.setLength(10);
      System.out.println(sb.length());
   }
}
```

You had to select 1 option

- It will print 5.
    - Although it truncates the string to length 5 but setLength(10) will append 5 spaces (actually null chars i.e. \u0000).
- It will print 10.
- It will print 8.
- Compilation error.
- None of the above.
    - The program will compile without error and will print 10 when run.

Explanation

If you do `System.out.println(sb);` it will indeed print "12345 " (without quotes) but the length will be 10.

From javadocs:

`public void setLength(int newLength)`

Sets the length of the character sequence. The sequence is changed to a new character sequence whose length is specified by the argument. For every nonnegative index k less than newLength, the character at index k in the new character sequence is the same as the character at index k in the old sequence if k is less than the length of the old character sequence; otherwise, it is the null character '\u0000'. In other words, if the newLength argument is less than the current length, the length is changed to the specified length.

If the newLength argument is greater than or equal to the current length, sufficient null characters ('\u0000') are appended so that length becomes the newLength argument.

The newLength argument must be greater than or equal to 0.

Parameters:

`newLength` - the new length

Throws:

`IndexOutOfBoundsException` - if the newLength argument is negative.

Problema 17 - LocalTime

Identify the correct statements.

You had to select 1 option

- LocalDate, LocalTime, and LocalDateTime extend Date.
- LocalDate, LocalTime, and LocalDateTime implement TemporalAccessor.
- Both - LocalDate and LocalTime extend LocalDateTime, which extends java.util.Date.
- LocalDate, LocalTime, and LocalDateTime implement TemporalAccessor and extend java.util.Date.

Explanation:

Here are some points that you should keep in mind about the new Date/Time classes introduced in Java 8 -

1. They are in package `java.time` and they have no relation at all to the old `java.util.Date` and `java.sql.Date`.

2. `java.time.temporal.TemporalAccessor` is the base interface that is implemented by `LocalDate, LocalTime,` and `LocalDateTime` concrete classes. This interface defines read-only access to temporal objects, such as a date, time, offset or some combination of these, which are represented by the interface `TemporalField`.

3. `LocalDate, LocalTime,` and `LocalDateTime` classes do not have any parent/child relationship among themselves. As their names imply, `LocalDate` contains just the date information and no time information, `LocalTime` contains only time and no date, while `LocalDateTime` contains date as well as time. None of them contains zone information. For that, you can use `ZonedDateTime`.

These classes are immutable and have no public constructors. You create objects of these classes using their static factory methods such as `of(...)` and `from(TemporalAccessor ).` For example, `LocalDate ld = LocalDate.of(2015, Month.JANUARY, 1);` or `LocalDate ld = LocalDate.from(anotherDate);` or `LocalDateTime ldt = LocalDateTime.of(2015, Month.JANUARY, 1, 21, 10); //9.10 PM`

Since you can't modify them once created, if you want to create new object with some changes to the original, you can use the instance method named `with(...)`. For example,

```
LocalDate sunday =
      ld.with(java.time.temporal.TemporalAdjusters.next(DayOfWeek.SUNDAY));
```

4. Formatting of date objects into String and parsing of Strings into date objects is done by `java.time.format.DateTimeFormatter` class. This class provides public static references to readymade DateTimeFormatter objects through the fields named ISO_DATE, ISO_LOCAL_DATE, ISO_LOCAL_DATE_TIME, etc. For example -

```
LocalDate d1 =
      LocalDate.parse("2015-01-01", DateTimeFormatter.ISO_LOCAL_DATE);
```

The parameter type and return type of the methods of DateTimeFormatter class is the base interface `TemporalAccessor` instead of concrete classes such as `LocalDate` or `LocalDateTime`. So you shouldn't directly cast the returned values to concrete classes like this -

```
LocalDate d2 =
(LocalDate) DateTimeFormatter.ISO_LOCAL_DATE.parse("2015-01-01"); //this
will compile
//but may or may not throw a ClassCastException at runtime.
```

You should do like this -

```
 LocalDate d2 =
    LocalDate.from(DateTimeFormatter.ISO_LOCAL_DATE.parse("2015-01-01"));
```

5. Besides dates, `java.time` package also provides `Period` and `Duration` classes. `Period` is used for quantity or amount of time in terms of years, months and days, while `Duration` is used for quantity or amount of time in terms of hour, minute, and seconds.

Durations and periods differ in their treatment of daylight savings time when added to ZonedDateTime. A Duration will add an exact number of seconds, thus a duration of one day is always exactly 24 hours. By contrast, a Period will add a conceptual day, trying to maintain the local time.

For example, consider adding a period of one day and a duration of one day to 18:00 on the evening before a daylight savings gap. The Period will add the conceptual day and result in a ZonedDateTime at 18:00 the following day. By contrast, the Duration will add exactly 24 hours, resulting in a ZonedDateTime at 19:00 the following day (assuming a one hour DST gap).

Problema 18 - fullPhoneNumber V2

Assuming that the following method will always be called with a phone number in the format ddd-ddd-dddd (where d stands for a digit), what can be inserted at //1 so that it will return a String containing "xxx-xxx-"+dddd, where dddd represents the same four digits in the original number?

```
public static String hidePhone(String fullPhoneNumber){
  //1 Insert code here
}
```

You had to select 2 options

- ```
  String mask = "xxx-xxx-";
  mask.append(fullPhoneNumber.substring(8));
  return mask;
  ```
  - Remember that String class doesn't have append (and insert) method because a String cannot be mutated.
- `return new StringBuilder("xxx-xxx-")+fullPhoneNumber.substring(8);`
- `return new StringBuilder(fullPhoneNumber).replace(0, 7, "xxx-xxx-").toString();`
  - For all of the methods in String and StringBuilder that take two int parameters for specifying a range, remember that the first index is included but the last index is not.
  - For example, as in this case, the arguments given are 0 and 7, which means it will include the characters with index 0 to 6, that is, a total of 7 characters 0, 1, 2, 3, 4, 5, and 6. Therefore, this will actually produce "xxx-xxx--dddd".
  - The same pattern is used for almost all other methods in standard java library classes. The first index is included but the last one is not.
- `return "xxx-xxx-"+fullPhoneNumber.substring(8, 12);`
  - This is another example where the pattern discussed above is used. The character at first index i.e. 8 is included but the last index 12 is not. In fact there is no element at the 12th index in the given string. So the characters returns by the substring will be the ones at index 8, 9, 10, and 11 of the original fullPhoneNumber.
  ```

Problema 19 - Tuesday

You want to print the date that represents upcoming tuesday from now even if the current day is a tuesday. Which of the following lines of code accomplishe(s) this?

You had to select 2 options

- `System.out.println(LocalDate.now()`
  `.with(TemporalAdjusters.next(DayOfWeek.TUESDAY)));`
- `System.out.println(LocalDate.now()`
  `.with(TemporalAdjusters.nextOrSame(DayOfWeek.TUESDAY)));`
  - This will return today's date if it is a tuesday, which is not what the question wants.
- `System.out.println(new LocalDate()`
  `.with(TemporalAdjusters.next(DayOfWeek.TUESDAY)));`
  - You cannot create a LocalDate object using its constructor because it is private.
- `System.out.println(new LocalDate()`
  `.adjust(TemporalAdjusters.next(DayOfWeek.TUESDAY)));`
  - adjust is not a valid method in LocalDate.
- `System.out.println(TemporalAdjusters.next(DayOfWeek.TUESDAY)`
  `.adjustInto(LocalDate.now()));`

Explanation

The JavaDoc description of `java.time.temporal.TemporalAdjusters` is very helpful:

Adjusters are a key tool for modifying temporal objects. They exist to externalize the process of adjustment, permitting different approaches, as per the strategy design pattern. Examples might be an adjuster that sets the date avoiding weekends, or one that sets the date to the last day of the month. There are two equivalent ways of using a TemporalAdjuster. The first is to invoke the method on the interface directly. The second is to use Temporal.with(TemporalAdjuster):

```
// these two lines are equivalent, but the second approach is recommended
temporal = thisAdjuster.adjustInto(temporal);
temporal = temporal.with(thisAdjuster);
```

It is recommended to use the second approach, `with(TemporalAdjuster)`, as it is a lot clearer to read in code.
This class contains a standard set of adjusters, available as static methods. These include:
finding the first or last day of the month
finding the first day of next month
finding the first or last day of the year
finding the first day of next year
finding the first or last day-of-week within a month, such as "first Wednesday in June"
finding the next or previous day-of-week, such as "next Thursday"

Problema 20 - ‘2015-02-05’

Given:

```
LocalDate d1 = LocalDate.parse("2015-02-05", DateTimeFormatter.ISO_DATE);
LocalDate d2 = LocalDate.of(2015, 2, 5);
LocalDate d3 = LocalDate.now();
System.out.println(d1);
System.out.println(d2);
System.out.println(d3);
```

Assuming that the current date on the system is 5th Feb, 2015, which of the following will be a part of the output?

You had to select 1 option

- 5th Feb, 2015
- 2015-02-05T00:00:00
    - Since LocalDate is being created (and not LocalDateTime), none of the printlns will output the time component.
- 02/05/2015
- 05/02/2015
- java.time.format.DateTimeParseException
- None of the above.

Explanation:

All the three printlns will produce `2015-02-05`.

Problema 21 - String Object

Which of the following methods can be called on a String object?

You had to select 3 options

- `substring(int i)`
  - returns substring starting from i to end.
- `substring(int i, int j)`
  - returns substring starting from i to j-1.
- `substring(int i, int j, int k)`
- `equals(Object o)`
  - Since Object class has this method, every java class inherits it.

Problema 22 - Print

What will be written to the standard output when the following program is run?

```
public class TrimTest{
   public static void main(String args[]){
      String blank  = " ";  // one space
      String line = blank + "hello" + blank + blank;
      line.concat("world");
      String newLine  =  line.trim();
      System.out.println((int)(line.length() + newLine.length()));
   }
}
```

You had to select 1 option

- 25
- 24
- 23
- 22
- None of the above.
  - It will print 13 !!!

Explanation

e that `line.concat("world")` does not change line itself. It creates a new String object containing `" hello  world"` but it is lost because there is no reference to it.
Similarly, calling `trim()` does not change the object itself.
So the answer is `8 + 5 = 13` !

Problema 23 - Hello

What will the following code print?

```
public class Test{
    public static void stringTest(String s){
        s.replace('h', 's');
    }
    public static void stringBuilderTest(StringBuilder s){
        s.append("o");
    }
    public static void main(String[] args){
        String s = "hell";
        StringBuilder sb = new StringBuilder("well");
        stringTest(s);
        stringBuilderTest(sb);
        System.out.println(s + sb);
    }
}
```

You had to select 1 option

- sellwello
- hellwello
- hellwell
- sellwell
- None of these.

Explanation

A String is immutable while a StringBuilder is not. So in `stringTest()`, `"hell".replace('h', 's')` will produce a new String `"sell"` but will not affect the original String that was passed to the method.
However, the `append()` method of `StringBuilder` appends to the original String object. So, `"well"` becomes `"wello"`.

Problema 24 - Blooper Whopper

What will the following code print when compiled and run?

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

    String s = "blooper";
    StringBuilder sb = new StringBuilder(s);
    s.append("whopper");
    sb.append("shopper");

    System.out.println(s);
    System.out.println(sb);
  }
}
```

You had to select 1 option

- blooper and bloopershopper
- blooperwhopper and bloopershopper
- blooper and blooperwhoppershopper
- It will not compile.
    - append() method does not exist in String class. It exits only in StringBuffer and StringBuilder. The value of sb will be bloopershopper though.

Problema 25 - ArrayList

Identify the correct statements about ArrayList?

You had to select 3 options

- ArrayList extends `java.util.AbstractList`.
    - ArrayList is a subclass of AbstractList.
      ```
      java.lang.Object
       -  java.util.AbstractCollection<E>
         -    java.util.AbstractList<E>
           -      java.util.ArrayList<E>
      ```
    - All Implemented Interfaces:
      `Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess`
- It allows you to access its elements in random order.
    - This is true because you can directly access any element using get(index) method. (This is unlike a LinkedList, in which you have to go through all the elements occuring before Nth element before you can access the Nth element.)
- You must specify the class of objects you want to store in ArrayList when you declare a variable of type ArrayList.
    - This is not true because you can still use non-generic form. For example, instead of using ArrayList<String> listOfStrings; you can use: ArrayList listOfStrings; Of course, if you use non generic version, you will lose the compile time type checking.

- ArrayList does not implement RandomAccess.
    - It does. RandomAccess is a marker interface used by List implementations to indicate that they support fast (generally constant time) random access. The primary purpose of this interface is to allow generic algorithms to alter their behavior to provide good performance when applied to either random or sequential access lists.
- You can sort its elements using Collections.sort() method.
    - An ArrayList is a List so you can use it where ever a List is required. This include Collections methods such as sort, reverse, and shuffle.

Problema 26 - Snorkel Yoodler

// Given:

StringBuilder b1 = new StringBuilder("snorkler");
StringBuilder b2 = new StringBuilder("yoodler");
// Write the contents of b1 and b2 after the statements shown on the left are
b1.append(b2.substring(2, 5).toUpperCase());
b2.insert(3, b1.append("a"));
b1.replace(3, 4, b2.substring(4)).append(b2.append(false));

Statements	Contents of b1	Contents of b2
`b1.append(b2.substring(2, 5).toUpperCase());`	snorklerODL	yoodler
`b2.insert(3, b1.append("a"));`	snorklera	yoosnorkleradler
`b1.replace(3, 4, b2.substring(4)).append(b2.append(false));`	snolerkleryoodlerfalse	yoodlerfalse

Explanation

You need to understand how `append, insert, delete,` and `substring` methods of `StringBuilder/StringBuffer` work. Please go through JavaDoc API for these methods. This is very important for the exam. Observe that `substring()` does not modify the object it is invoked on but `append, insert` and `delete` do.

In the exam, you will find questions that use such quirky syntax, where multiple calls are chained together. For example: `sb.append("a").append("asdf").insert(2, "asdf")`. Make yourself familiar with this technique. If in doubt, just break it down into multiple calls. For example, the aforementioned statement can be thought of as:

```
sb.append("a");
sb.append("asdf");
sb.insert(2, "asdf")
```

Note that the method `substring()` in `StringBuilder/StringBuffer` returns a `String` (and not a reference to itself, unlike `append, insert,` and `delete`). So another StringBuilder method cannot be chained to it. For example, the following is not valid: `sb.append("a").substring(0, 4).insert(2, "asdf");`

The following is valid though: `String str = sb.append("a").insert(2, "asdf").substring(0, 4);` 'substring' () de StringBuilder regresa un `String`

Problema 27

Which of these are not part of the StringBuilder class?

You had to select 1 option

- `trim( )`
    - This method is in String class.
- `ensureCapacity(int )`
    - Ensures that the capacity of the buffer is at least equal to the specified minimum.
- `append(boolean)`
    - It has all sorts of overloaded append methods !!!
- `reverse( )`
- `setLength(int)`
    - Sets the length of the underlying string buffer. This string buffer is altered to represent a new character sequence whose length is specified by the argument. For every nonnegative index k less than newLength, the character at index k in the new character sequence is the same as the character at index k in the old sequence if k is less than the length of the old character sequence; otherwise, it is the null character '' (\u0000). In other words, if the newLength argument is less than the current length of the string buffer, the string buffer is truncated to contain exactly the number of characters given by the newLength argument.
    - If the newLength argument is greater than or equal to the current length, sufficient null characters ('\u0000') are appended to the string buffer so that length becomes the newLength argument.
    - The newLength argument must be greater than or equal to 0.
      Parameters:
      newLength - the new length of the buffer.
      Throws:
      IndexOutOfBoundsException - if the newLength argument is negative.

Problema 28

Which of the following statements are true?

You had to select 2 options

- method length() of String class is a final method.
    - Actually, String class itself is final and so all of its methods are implicitly final.
- You can make mutable subclasses of the String class.
    - Both - String and StringBuilder are final classes. So is StringBuffer.
- StringBuilder extends String.
    - StringBuilder extends Object
- StringBuilder is a final class.
    - String, StringBuilder, and StringBuffer - all are final classes.
    - 1. Remember that wrapper classes (java.lang.Boolean, java.lang.Integer, java.lang.Long, java.lang.Short etc.) are also final and so they cannot be extended.
    - 2. java.lang.Number, however, is not final. Integer, Long, Double etc. extend Number.
    - 3. java.lang.System is final as well.
- String class is not final.

Problema 29

Which of the following are true regarding the new Date-Time API of Java 8?
You had to select 2 options

- It uses the calendar system defined in ISO-8601 as the default calendar.
    - This calendar is based on the Gregorian calendar system and is used globally as the defacto standard for representing date and time. The core classes in the Date-Time API have names such as LocalDateTime, ZonedDateTime, and OffsetDateTime. All of these use the ISO calendar system.
    - If you want to use an alternative calendar system, such as Hijrah or Thai Buddhist, the java.time.chrono package allows you to use one of the predefined calendar systems. Or you can create your own.
- Most of the actual date related classes in the Date-Time API such as LocalDate, LocalTime, and LocalDateTime are immutable.
    - These classes do not have any setters. Once created you cannot change their contents. Even their constructors are private.
- LocalDateTime includes time zone information but LocalDate does not.
    - None of LocalDate, LocalDateTime, or LocalTime store zone information.
    - `java.time.ZonedDateTime` does. ZonedDateTime is an immutable representation of a date-time with a time-zone. This class stores all date and time fields, to a precision of nanoseconds, and a time-zone, with a zone offset used to handle ambiguous local date-times. For example, the value "2nd October 2007 at 13:45.30.123456789 +02:00 in the Europe/Paris time-zone" can be stored in a ZonedDateTime.
    - ZonedDateTime is not listed in official exam objectives.
- To create a LocalDate or a LocalDateTime object, you can use one of their several constructors.
    - These classes do not have any public constructors. You need to use their static factory methods to get their instances. For example:
    - java.time.LocalDate d1 = java.time.LocalDate.of(2015, Month.JANUARY, 31);
    - java.time.LocalDateTime d2 =
      java.time.LocalDateTime.of(2015, Month.JANUARY, 31, 10, 56);
    - java.time.LocalDateTime d3 = \
      java.time.LocalDateTime.parse("2015-01-02T17:13:50"); //Note that this
      //will throw a java.time.format.DateTimeParseException if the input string
      //lacks the time component i.e.T17:13:50
    - java.time.LocalDate d4 = java.time.LocalDate.parse("2015-01-02"); //Note that
      //this will throw a java.time.format.DateTimeParseException if the input string
    - //contains the time component java.time.LocalTime d5 =
      java.time.LocalTime.parse("02:13:59.985"); //Note that
      //this will throw a java.time.format.DateTimeParseException if the input string
      //contains the Date component

Problema 30 - ‘g’ ‘g’

Which of the following statements will evaluate to true?

You had to select 1 option

- `"String".replace('g','G') == "String".replace('g','G')`
    - replace creates a new string object.
- `"String".replace('g','g') == new String("String").replace('g','g')`
- `"String".replace('g','G')=="StrinG"`
    - replace creates a new string object.
- `"String".replace('g','g')=="String"`
    - replace returns the same object if there is no change.
- None of these.

Explanation

There are 2 points to remember:

1. `replace(char oldChar, char newChar)` method returns the same `String` object if both the parameters are same, i.e. if there is no change. Thus, `"String" == "String".replace('g', 'g')` will return `true`.
However, the JavaDoc API reference for this method does not clearly specify this behavior and so, it would not be a good idea to rely upon it.

2. `replace(CharSequence oldSeq, CharSequence newSeq)` method may or may not return a new `String` object even if there is no change after replacement. For example, `"String" == "String".replace("g", "g")` return `true` but `"String" == "String".replace("ng", "ng")` returns `false`. This behavior is also not clearly specified by the JavaDoc.

Problema 31

Consider following classes:

```
//In File Other.java
package other;
public class Other { public static String hello = "Hello"; }

//In File Test.java
package testPackage;
import other.*;
class Test{
   public static void main(String[] args){
      String hello = "Hello", lo = "lo";
      System.out.print((testPackage.Other.hello == hello) + " ");    //line
1
      System.out.print((other.Other.hello == hello) + " ");   //line 2
      System.out.print((hello == ("Hel"+"lo")) + " ");           //line 3
      System.out.print((hello == ("Hel"+lo)) + " ");              //line 4
      System.out.println(hello == ("Hel"+lo).intern());          //line 5
   }
}
class Other { static String hello = "Hello"; }
```

What will be the output of running class Test?

You had to select 1 option

- false false true false true
- false true true false true
- true true true true true
- true true true false true
- None of the above.

Explanation

These are the six facts on Strings:

1. Literal strings within the same class in the same package represent references to the same String object.
2. Literal strings within different classes in the same package represent references to the same String object.
3. Literal strings within different classes in different packages likewise represent references to the same String object.
4. Strings computed by constant expressions are computed at compile time and then treated as if they were literals.
5. Strings computed at run time are newly created and therefore are distinct. (So line 4 prints false.)
6. The result of explicitly interning a computed string is the same string as any pre-existing literal string with the same contents. (So line 5 prints true.)

Problema 32

What will the following statement return?

```
"    hello java guru   ".trim();
```

You had to select 1 option

- The line of code will not compile.
    - " hello java guru " is a valid String and trim() is a valid method in String class.
- "hellojavaguru"
    - trim() does not remove spaces in within the string but the spaces at the beginning and at the end.
- "hello java guru"
- "hello java guru "
    - It returns a string in which both the leading and trailing white space of the original string are removed.
- None of the above

Problema 33

Which of the following operators can be used in conjunction with a String object?

You had to select 3 options

- `+`
- `++`
- `+=`
- `.`
- `*`

Explanation
Only + is overloaded for String. `a+=x` is actually converted to `a = a + x`. so it is valid for Strings. dot (.) operator accesses members of the String object. There is only one member variable though: CASE_INSENSITIVE_ORDER. It is of type Comparator (which is an interface).

Problema 34

Consider the following code:

```
public class Logger{
    private StringBuilder sb = new StringBuilder();

    public void logMsg(String location, String message){
    sb.append(location);
    sb.append("-");
    sb.append(message);
    }

    public void dumpLog(){
    System.out.println(sb.toString());
    //Empty the contents of sb here
    }

}
```

Which of the following options will empty the contents of the StringBuilder referred to by variable sb in method dumpLog()?

You had to select 1 option

- sb.delete(0, sb.length());
- sb.clear();
- sb.empty();
- sb.removeAll();
- sb.deleteAll();

Explanation
`public StringBuilder delete(int start, int end)`
Removes the characters in a substring of this sequence. The substring begins at the specified start and extends to the character at index end - 1 or to the end of the sequence if no such character exists. If start is equal to end, no changes are made.
Parameters:
start - The beginning index, inclusive.
end - The ending index, exclusive.
Returns:
This object.
Throws:
StringIndexOutOfBoundsException - if start is negative, greater than length(), or greater than end.

Problema 35

What will the following line of code print?

```
System.out.println(LocalDate.of(2015, Month.JANUARY, 01)
    .format(DateTimeFormatter.ISO_DATE_TIME));
```

You had to select 1 option

- `01 Jan 2015`
- `01 January 2015 00:00:00`
- `2015-01-01`
- `2015-01-01T00:00:00`
- Exception at run time.
    - Observe that you are creating a `LocalDate` and not a `LocalDateTime`. `LocalDate` doesn't have time component and therefore, you cannot format it with a formatter that expects time component such as `DateTimeFormatter.ISO_DATE_TIME`.
    - Thus, it will print `java.time.temporal.UnsupportedTemporalTypeException: Unsupported field: HourOfDay exception` message.
    - If you use `DateTimeFormatter.ISO_DATE`, it will print `2015-01-01`
    - Also, remember that a `LocalDateTime` object can be formatted using a `DateTimeFormatter.ISO_DATE` though.

Problema 36

What will the following lines of code print

```
java.time.LocalDate dt = java.time.LocalDate.parse("2015-01-01")
     .minusMonths(1).minusDays(1).plusYears(1);
System.out.println(dt);
```

You had to select 1 option

- Compilation error.
- Exception at run time.
- `2015-12-31`
- `2015-11-30`
    - The numbering for days and months starts with 1. Rest is simple math.

Explanation

Observe that most of the methods of LocalDate (as well as LocalTime and LocalDateTime) return an object of the same class. This allows you to chain the calls as done in this question. However, these methods return a new object. They don't modify the object on which the method is called.