Emre KARAGÖZ

ASSIGNMENT 10
Java Practices

Q01-) Option E

The code compiles and throws "`StringIndexOutOfBoundsException`" at runtime. "`sb.indexOf("c")`" causes this exception because "`sb`" is empty.

Q02-) Option C & Option E

+= (level 1), ++ (level 13 or level 14**)**

Option A: – (level 11), + (level 11), = (level 1), -- (level 13 or level 14)
Option B: % (level 12),* (level 12),/ (level 12),+ (level 11)
Option C: = (level 1), + (level 11), / (level 12), * (level 12)
Option D: ^ (level 6), * (level 12), - (level 11), == (level 8)
Option E: * (level 12), / (level 12), % (level 12), -- (level 13 or level 14)

** http://www.cs.bilkent.edu.tr/~guvenir/courses/CS101/op_precedence.html

Q03-) Option B & Option C & Option F

Java Beans Naming Conventions

Rule	Example
Properties are private.	`private int numEggs;`

Getter methods begin with is if the property is a boolean.	`public boolean isHappy() {`
	`  return happy;`
	`}`

Getter methods begin with get if the property is not a boolean.	`public int getNumEggs() {`
	`  return numEggs;`
	`}`

Setter methods begin with set.	`public void setHappy(boolean happy) {`
	`  this.happy = happy;`
	`}`

The method name must have a prefix of set/get/is, followed by the first letter of the property in uppercase, followed by the rest of the property name.	`public void setNumEggs(int num) {`
	`  numEggs = num;`
	`}`

Option A: `public byte getNose(String nose)`

A getter should not take an argument.

Option B: `public void setHead(int head)`

Option C: `public String getShoulders()`

Option D: `public long isMouth()`

Only a getter which return boolean should begin is.

Option E: `public void gimmeEars()`

`"gimme" is not a` JavaBeans `prefix`

Option F: `public boolean isToes()`

Options B, C and E are valid JavaBeans signatures.

Q04-) Option A & Option E

The code does not compile due to line 24. In order to get number of elements of an array, "length" property should be used. "size()" prevents the code from compiling. If we fix this error and compile the code, "Integer.valueOf('x')" is assigned to only half of the elements. The other elements are equals to "0" which is is the initial value.

Q05-) Option B & Option D

Entering an invalid input and not being able to access the file system are can be recovered. Throwing an Error is not proper for these cases. So, Option A and Option E are incorrect. Error is subclass of Throwable. This makes Option C is incorrect. Even though an Error can be caught, this is not recommended because Errors are fatal. Options B and D are correct.

Q06-) Option A & Option C & Option E

- `import forest.Bird;`

  Allows `forest.Bird` **(Option A)**

- `import jungle.tree.*;`

  Allows `jungle.tree.Huicungo` **(Option C)**

- `import savana.*;`

Option B: `savana.sand.Wave`

`"import savana.sand.*";` is required

Option D: `java.lang.Object`

All Java class imports `java.lang.*`

Option E: `forest.Sloth`

`"import forest.Sloth"` or `" import forest.*"` are required.

Option F: `forest.ape.bonobo`

`"import forest.ape.bonobo"` or `" import forest.ape*"` are required.

Q07-) Option C

```
ArrayList l = new ArrayList();
String s = new String();
StringBuilder sb = new StringBuilder();
LocalDateTime t = LocalDateTime.now();
```

`ArrayList, StringBuilder` are mutable while `String` and `LocalDateTime` are `immutable.` **Option C** is the answer.

Q08-) Option C

The code compiles and prints `"wing".` At first iteration the first five character `("Leave")` are deleted. The loop iterates second time because still there are more than five character. At the second iteration, the second five characters `(" s gro")` are deleted and output becomes `"wing".`

Q09-) Option D

The code compiles and prints `"false false true".` Three "String" objects are created here. Even though same value is assigned them, they are completely different objects. So `"=="` operator returns `"false".` Since they have same value, `"equals()"` method returns true. Because it compares the values. **Option D** is the answer.

Q10-) Option C

"==" operator compares the objects rather than their values. So, lines 5 and 6 prints false. Line 7 also prints false because "LOL" and "`lol`" are different from each other. Since they compare values "`equals()`" and "`equalsIgnoreCase()`" methods, the other 3 line print "`true`". **Option C** is the answer.

Q11-) Option A & Option B & Option C

Line 15 and line 17 can be removed because the label "outer" and "inner" are not referenced. Since the inner loop has broader condition than the outer loops condition, the outer do while also can be removed (line 16 and line 21). This does not affect the output. The options they consist these lines are correct. So, Option A, B and C are the answer.

Q12-) Option B & Option C

Options C and B prevent the code from compiling because a "long" variable cannot contain a decimal point.

Q13-) Option A

The code prints nothing. "time" is "01:11" and "time.getHour()" returns "1". Since the condition is false, the loop is never entered. **Option A** is the answer.

Q14-) Option D

Since "`game`" is not initialized, it points to null which is the default value of class variables. The code compiles but it throws a "`NullPointerException`" at runtime. **Option D** is the answer.

Q15-) Option C & Option E

These are the correct syntax of creation of an ArrayList.

```
ArrayList<String> list4 = new ArrayList<String>();
ArrayList<String> list5 = new ArrayList<>();
```

Options B, D, and F prevents the code from compiling because identifier is expected at the left side of the "=". Option A gives a warning that tells "raw use of parameterized class 'ArrayList'.

Q16-) Option B & Option D

At the end, "`flip flop`" is pointed by "`shoe1`" and "`shoe3`", "`croc`" is pointed by "`shoe2`". But, none of these objects points to "`sandal`". So, it is eligible for garbage collection. Nevertheless, garbage collection is not guaranteed to run.

Q17-) Option C

Calling the "`getFish`" method in the main method prevents the code from compiling. It is an overridden method so it may throw "`BubbleException`" in addition to "`RuntimeException`". On order to compile this code, these exceptions must be handled. **Option C** is the answer.

Q18-) Option A

The code compiles and prints "-5". "`ArrayList list`" is created with elements "-5", "0", and "5". Then, "`print()`" method controls the elements and prints which are less than zero. **Option A** is the answer.

Q19-) Option F

"catch" and ***"finally"*** keywords are required with a try statement. They can be used both but, one of them is sufficient to compile the code. So they are not mandatory. ***"finalize"*** is not a keyword. ***"finalize()"*** is the method of Object class. This method is called just before an object is garbage collected. finalize() method overrides to dispose system resources, perform clean-up activities and minimize memory leaks. "throws" keyword is used to declare exceptions. **Option F** is the answer.

Q20-) Option A

The code compiles and prints "5". At the beginning result is "8". Since condition is true, "`loop`" is entered and "`result`" is incremented by "1". Then, it is decremented by "1" until it becomes "5". After this, "`loop`" is skipped and "`result`" is printed. **Option A** is the answer.

Q21-) Option C

The code compiles and prints "1 2". "`static int teeth`" is set to "0" which is the default value. Then, it is incremented by 1 by constructor and "`snap()`" method prints its value. After this, its value is incremented by "1" by constructor again and becomes "2" and it is printed. Notice that, decrementing in snap method does not affect the "`static int teeth`". In this method, its argument "`int teeth`" is decremented. **Option C** is the answer.

Q22-) Option A

The code compiles and prints "b" which is the value of "`String witch`". Since "`String`" is an immutable class, result of "`String.concat()`" is ignored. It does not change the value of "`witch`". **Option A** is the answer.

Q23-) Option A & Option F

All top-level interfaces are assumed to have **public** or **default (Option A)** access, and they must include the **abstract (Option F)** modifier in their definition. Therefore, marking an interface as **private (Option D)** , **protected (Option B)**, or **final (Option E)** will trigger a compiler error, since this is incompatible with these assumptions.

Q24-) Option D

The code does not compile due to the condition statement of while loop. "`tie=null`" does not return a boolean value. In order to compile this code, this line corrected as "`tie==null`". If we do this, it compiles and prints "`shoelace`". **Option D** is the answer.

Q25-) Option B & Option F

If a file does not contain a package statement, compiler considers the class part of the default package. So, Java does not require every file to declare a package statement. This makes Option A and D incorrect. Moreover the package "java.lang" is imported into every Java class automatically without an import statement. So **Option B** is correct and Option C and E are incorrect. **Option F** is also correct. If the class declaration does not extend another class, then it implicitly extends the java.lang.Object class.

Q26-) Option D

The code does not compile due to line "`m1`". Because, Java does not allow a class to inherit two interfaces that declare the same default method, unless the class overrides that methods them. **Option D** is the answer.

Q27-) Option C

The code does not compile because variable "profit$$$" is never assigned. **Option C** is the answer.

Q28-) Option A

Given a variable x, "**`x--`**" decreases the value of x by 1 and returns the original value, while "**`++x`**" increases the value of x by 1 and returns the new value. **Option A** is the answer.

Q29-) –

Q30-) Option E & Option F

In order to be accessible from the "`main()`" method, "`max`" and "`min`" must be "`static`". Because non-static field "`max`" and "`min`" cannot be referenced from a static context. **Option E** and **Option F** are correct.

Q31-) Option B & Option E

Option A is incorrect because the ternary operator only evaluates one of the two right-hand expressions at runtime. If the condition is true, the left one is evaluated. Otherwise, the right one is evaluated. Since a switch statement may contain at most one default statement, **Option B** is correct. A single if-then statement cannot have more than one else statements, **Option C** is also incorrect. The "|" and "||" operator are not interchangeable. The short-circuit operator "||" is nearly identical to the operator "|" except that the right-hand side of the expression may never be evaluated if the final result can be determined by the left-hand side of the expression. So **Option D** is incorrect. Sinc the "!" operator cannot be applied to numeric expressions, **Option E** is also correct.

Q32-) Option C
The code compiles and prints "ed." An empty string builder sb is created and "red" is added it by "append()" method. "delete(0)" deletes the first character of "sb". Since there is not an index which starts from index one end ends right before index 0, "delete(1,1)" removes nothing. **Option C** is the answer.

Q33-) Option A & Option D

A method name may only contain letters, numbers, $, or _. Also, the first character is not allowed to be a number, and reserved words are not allowed. By convention, methods begin with a lowercase letter but are not required to. "`_____()`" **(Option A)** and "`$Hum2()`" **(Option D)** are valid method names in Java.

Q34-) -

Q35-) Option D

I. The java command uses . to separate packages.
II. Java supports functional programming.
III. Java is object oriented.
IV. Java supports polymorphism.

All these statements are true. Notice that, Java supports polymorphism by Lambda expressions. **Option E** is the answer.

Q36-) Option D

Since "`String listing`" has 3 elements and each these elements has two other strings, the code compiles and prints "`3 2`".

Q37-) Option A & Option B and Option A

Data types supported by switch statements include the following:

- `int` and `Integer`
- `byte` and `Byte`
- `short` and `Short`
- `char` and `Character`
- `int` and `Integer`
- `String`
- `enum` values

Types that includes floating-point such as float and double and their wrapper classes are not supported.

Q38-) Option D

The code does not compile due to line "k2" because the syntax of lambda expression is wrong. "->" must be used instead of "=>". If we fix this error, the code compiles and prints "`true`". **Option D** is the answer.

Q39-) Option B & Option C and Option E

There are three types of comments in Java. The first is called a single-line comment. It begins with two slashes ("//"). Anything you type after that on the same line is ignored by the compiler. **Option C** is a single-line comment example. The other type of comment is multiple-line or multiline comment. A multiple-line comment includes anything starting from the symbol "/*" until the symbol "*/". The third type of comment is similar to a multiline comment except it starts with "/**". This special syntax tells the "Javadoc" tool to pay attention to the comment. Javadoc comments have a specific structure that the Javadoc tool knows how to read. **Option B and Option E** are examples of this type of comment.

Q40-) Option A & Option F

Import statements at **Option A** and **Option F** allow class "`Deer`" to compile because these statements import the static members of class "`Grass`" which are "`getGrass`" method and variable "`seeds`".

Q41-) Option D (Why this exception is thrown?)

The code compiles but it throws "UnsupportedOperationException".

Q42-) Option A & Option D

In Java, a variable name may start with and underscore and we can add underscores anywhere except at the beginning of a literal, the end of a literal, right before a decimal point, or right after a decimal point. So, changing "`name`" to "`_name`" and "`10017`" to "`10_0_17`" do not prevent the code from compiling. **Option A & Option D** are the answers.

Q43-) Option B

The code compiles and prints "`[0, 01, 1, 10]`". String values sort in alphabetic order, so, 0 sorts before 1. (Numbers sort before letters and uppercase sorts before lowercase.). **Option B** is the answer.

Q44-) Option D

Using the ***"public"*** and ***"static"*** modifiers together allows a variable to be accessed from any class, without requiring an instance variable. The public modifier allows access members in any class, while the static modifier allows access without an instance of the class. **Option D** is the answer.

Q45-) Option A

The code compiles and prints one line: "`OCA OCAOCA OCPOCP OCAOCP OCP`"
Since braces are not used, "`println`" method is outside of the loop and it is executed only one time. **Option D** is the answer.

Q46-) Option C & Option D

The "***javac"* command** is used to compile Java programs, it takes a .java file as input and produces bytecode (.class) . Following is the syntax of this command.

"> javac sample.java"

The **java command** is used to execute the bytecode of java. It takes byte code as input and runs it and produces the output. Following is the syntax of this command.

"> java sample"

Options C and D are true statements.

Q47-) Option C

```
Integer four = Integer.parseInt("4");
int three = Integer.parseInt("3");
```

Only these two lines of code compiles without an issue. `"parseInt"` method returns an `"int"`. An `"int"` and its associated wrapper class `("Integer")` can store this value.

Q48-) Option B & Option D & Option F

`"boolean value = 5;"` is an invalid declaration. This prevents from the code compiling. Only `"public int drive(double car)"` and `"public int drive(short car)"` methods return "3". If `"value"` is a `"float"` the first method is invoked. If `"value"` is a `"byte"` or a `"short",` the second method is invoked and "3" is printed.

Q49-) Option C

The code throws a runtime exception because there is not an entry point of this class. In a Java application, an entry point must contain a main method with access modifiers *"public"* and *"static"*, return type of *"void"* and a single *"String"* argument. The signature of the entry point should be like this:

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

"static", keyword is missing here. If we add this keyword, the code compiles and prints "true" 2 times. Conditions "bart" == "bart" and "3 == 3" are always "true". **Option C** is the answer.

Q50-) Option D

Even if static member "`Dance.swing(int…)`" accessed via instance reference this does not prevent from compiling. The code compiles and prints "`245`". First, an "`ArrayOutOfBoundsException`" is thrown due trying the third element of "`beats`" which does not exist. This exception is handled and "`2`" is printed. Then "`finally`" block is executed and prints "`4`". Finally, "`5`" is printed at main method and output becomes "`245`". **Option D** is the answer.

Q51-) Option E

The code compiles and prints "`cup,`", then it throws an "`ArrayIndexOutOfBoundsException`" at runtime. At the second iteration of for loop, "`drinks.get`" method attempts to access the third element of "`drinks`" which does not exist. "`drinks`" has two elements which are "`can`" and "`cup`". **Option E** is the answer.

Q52-) Option A & Option E & Option F

As explained in Q49; in a Java application, an entry point must contain a main method with access modifiers *"public"* and *"static"*, return type of *"void"* and a single *"String"* argument. The signature of the entry point should be like this:

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

Option A: `public static void main(String... widgets)`

Option B: `public static void main(String sprockets)`

- String cannot be an argument.

Option C: `protected static void main(String[] args)`

- An entry point must be public.

Option D: `public static int void main(String[] arg)`

- An entry point does not return a value.

Option E: `public static final void main(String []a)`
Option F: `public static void main(String[] data)`

Q53-) Option C & Option D & Option E

The code does not compile due to lines 10, 11, 12, 14. “winter” is type of long, it is not allowed in a switch statement. A switch statement can only have one “default” statement. Two “default” statements prevent the code from compiling. Finally, “fall” must be a final static variable.

Q54-) Option A & Option B & Option C

The code does not compile due to lines “h1”, “h2”, and “h3”. “h1” does compile because abstract interface methods can only be `"public"`. “h2” and “h3” do not compile because we can not reduce the visibility of inherited methods from interface `"Friend"`.

Q55-) Option B & Option E & Option F

Type	How to recognize	Okay for program to catch?	Is program required to handle or declare?
Runtime exception	Subclass of `RuntimeException`	Yes	No
Checked exception	Subclass of `Exception` but not subclass of `RuntimeException`	Yes	Yes
Error	Subclass of `Error`	No	No

Option A: `FileNotFoundException`

- A checked exception thrown programmatically when code tries to reference a file that does not exist.

Option B: `ArithmeticException`

- A unchecked exception thrown by the JVM when code attempts to divide by zero

Option C: `IOException`

- A checked exception thrown programmatically when there’s a problem reading or writing a file.

Option D: `Exception`

- The superclass of checked and unchecked exceptions.

Option E: `IllegalArgumentException`

- A unchecked exception thrown by the programmer to indicate that a method has been passed an illegal or inappropriate argument

Option F: `RuntimeException`

- A *runtime exception* is defined as the RuntimeException class and its subclasses.

Q56-) Option F

Since condition “space < 2” is always “false”, “ship” is always “10.0”. The first if statement and the else statement are executed and prints:

```
Goodbye
See you again
```

The code prints two lines, **Option F** is the answer.

Q57-) Option B & Option C & Option D

In order to compile this code, “`clock`” variable must be accessible from another class from same packege and, “`getClock()`” method must be accessible from another package. So, clock can be ***“package-private”***, ***“protected”*** or ***“public”***, while “`getClock()`” can be ***“protected”*** or ***“public”***.

Q58-) Option B

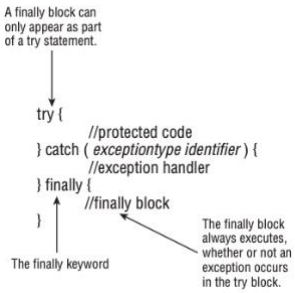
The code does no compile due tol ine “`q1`”. Class “`CarbonStructure`” must either be declared abstract or implement abstract method “`getCount()`” in “`CarbonStructure`”. **Option B** is the answer.

Q59-) Option C

The code does not compile due to two lines. Even though the signature of main method is not a proper signature to be an entry point, it is a valid method and it does not prevent the code from compiling. **Line 6** does not compile because “`LocalDate`” needs to use a static method such as “`LocalDate.of()`” rather than a constructor. There is not a class called “`DateTimeFormat`” in Java, so **line 8** does not compile. (“`DateTimeFormatter`” should be used in order to compile the code.)

Q60-) Option A & Option E

The syntax of a try statement with finally



"catch" and ***"finally"*** keywords are required with a try statement. They can be used both but, one of them is sufficient to compile the code. A try statement can have more than one catch and if there is a catch, finally block must appear after a catch block. A catch block cannot be appeared before a finally block. This prevents the code from compiling. **Options A and E** are true statements.

Q61-) Option B

Since condition "1 + 2 * 5>=2" is always "true", "`fish`" is equals to 4. Since condition "3 < 3" is always "false" the right side of the ternary operator is evaluated. Then, "`mammals`" becomes "9" because condition "5>=5" is always "true". The code compiles and prints "13". **Option B** is the answer.

Q62-) –

Q63-) Option F

Java also provides a convenient way to search—but only if the array is already sorted. The table below covers the rules for binary search.

Scenario	Result
Target element found in sorted array	Index of match
Target element not found in sorted array	Negative value showing one smaller than the negative of index, where a match needs to be inserted to preserve sorted order
Unsorted array	A surprise—this result isn't predictable

Since the "`args`" is not sorted, the output is not guaranteed. **Option F** is the answer.

Q64-) Option B

At the end of the program, “shoe1” points to “croc”, “shoe2” points to “sandal” and “shoe3” points to “croc”. Since “flip flop” is not referenced by any variable, it is eligible for garbage collection. **Option B** is the answer.

Q65-) Option E

The ***“throws”*** keyword is used in method declarations, the ***“finally”*** keyword is used to guarantee a statement will execute even if an exception is thrown, and the ***“throw”*** keyword is used to throw an exception to the surrounding process. **Option E** is the answer.

Q66-) Option B & Option E

The code prints following lines:

```
Downtown Day
Uptown Night
```

Then, an “ArrayIndexOutOfBoundsException” is thrown due to attempting to access the third element of “times” which does not exist. **Option B** and **E** are true statements.

Q67-) Option E

Because of *“**virtual methods**”*, it is possible to *“**override**”* a method, which allows Java to support *“**polymorphism**”*. Without virtual methods, method overriding would not be possible. It is the most important part of polymorphism.

Q68-) Option E

The code compiles but it throws an `"ArrayIndexOutOfBoundsException"` due to line “s2”. `"names[1]"` is “Fall”. `"Fall"` has 4 elements and this value is stored at variable “l”. Attempting to access `"names[4]"` couses this exception because names has only 4 elements. Fourth element of `"names"` is at index “3”. **Option E** is the answer.

Q69-) –

Q70-) Option B

The code compiles and prints `"6,LONG"`. `"adjustPropellers()"` returns “6” and this value is assigned to `"length"` variable at main method. The data in `"String[] type"` is modified and “LONG” is assigned to it. **Option B** is the answer.

Q71-) Option D

The code does not compile because of three lines. Since "`CableSnapException`" is a subclass of "`OpenDoorException`", the second catch block is an unreachable catch block. Since "ex" is already defined, another variable cannot have the same same. This is the second error. In addition, "`openDrawbridge()`" throws an "`Exception`". It must be declared in main method or handled.

Q72-) Option E

"Object orientation" and *"**encapsulation"*** are two properties that go hand in hand to improve class design by structuring a class with related attributes and actions while protecting the underlying data from access by other classes.

Q73-) Option E

The code does because not compile there is not a variable type such as "`string`" in JAVA. If we change "`string`" to "`String`", it compiles and prints "`false false`".

Q74-) Option A

The code compiles and prints "0". First, the code sorts "`args`" then it searches the index of the element at the index 0. So, it returns 0. **Option A** is the answer.

Q75-) Option B & Option C & Option D

Every for-each loop can be rewritten as a traditional for loop, as a while loop and as a do-while loop. So, options B, C, and D are true statements.

Q76-) Option E

"**`LocalDate`**" contains just a date, no time and no time zone. It does not support time, so it does not have methods such as "**`getHour()`**" and "**`plusHours()`**". Because of this, the code does not compile. **Option E** is the answer.

Q77-) Option C

Since both "`balls`" and "`scores`" set to "`null`", both of them are eligible for the Garbage Collection before the end of the main() method.

Q78-) Option B

The code compiles and prints “1”. Since braces are not used, “`count++`” is outside of the loop and it is executed only one time when loop is over and “`count`” becomes “1”. **Option B** is the answer.

Q79-) Option E

Four lines prevents the code from compiling. Line 10 does not compile because the override reduces the visibility of an inherited method. A method cannot have 2 return statements but, grunt() method has 2 return in this code(lines 12 and 13). Line 11 is causes an error because “`sing() += 10;`” is invalid. "`sing()`” is a method and it is not allowed there.

Q80-) Option B & Option E

The ***“default”*** keyword can optionally be used in a switch statement to label a block of statements to be executed if no case matches the specified value. Alternatively, the ***“default”*** keyword can be used to allow an interface to provide an implementation of a method. **Options B** and **E** are true.