Emre KARAGÖZ

ASSIGNMENT 1
JAVA BASICS
(HW 1)

Q1-) Option D

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 D: compensates these requirements (and an optional modifier *"final"* to prevent the method from to be overwritten) and it is the correct answer.

Option A: "static" access modifier is missing.

Option B: "String" argument is missing.

Option C: Access modifier and method name are wrong.

Q2-) Option A

Option B: This class diagram is an example of object-oriented design in Java so, the statement in this option is true.

Option C: According to this diagram, *"Metal"* class is the superclass (parent) and *"Gold"* and *"Silver"* classes are subclasses (child). In Java, subclasses inherit the attributes and methods from the superclass. So, The Gold and Silver classes inherit *"weight"* and *"color"* attributes from the Metal class and the statement in this option is true

Option D: *"Gold"* does not inherit the "luster" attribute because, "luster" attribute belongs to *"Silver"* class and *"Gold"* & *"Silver"* are both subclasses in this case. The statement in this option is true.

Option A: This is the correct answer. Platform independence means that when you compiled a Java code you can execute the program on any operating system. This has nothing to do with the diagram.

Q3-) Option A

In Java, ***".class"*** is the proper file extension for a Java bytecode compiled file. The correct answer is **Option A.**

Q4-) Option B

Even though "Date" class exist in both *"java.util"* and *"java.sql"* packages, we can import both packages. Namely, line 1 and line 2 compile with no error. Line 4 does not compile because of uncertainty of *"Date"* class we used. It may be from both packages, compiler cannot know which one it is and cannot create object *"rob"*. On the line 5, everything is certain. *"sharon"* object is created from Date class that exist in *"java.util"* package without issue. According to this information given above, statements in **Option A**, **Option C** and **Option D** are incorrect. Line 4 will not compile and this makes **Option B** is correct.

Q5-) Option A

A. Objects are grouped as procedures, separate from the data they act on.

In object-oriented programming languages casting means taking an object of one particular type and turning it into another object type. This process is called **casting** a variable. So, an object can take many forms via casting and statement in **Option B** is true. Classes have two primary elements: *methods* and *fields*. Variables hold the state of the program, methods operate on that state and an object can hold data. Namely, **Option C** and **Option D** are also correct. But objects are not grouped as procedures, separate from the data they act on. Object oriented programming tends to group data in individual object. **Option A** is incorrect, it is the answer.

Q6-) Option D

In Java, a local variable is a variable defined within a method and local variables must be initialized before use. Local variables can never have a scope larger than the method they are defined in. They have a limited scope and this makes **Option D** the correct answer.

Q7-) Option B

There's one special package in the Java world called *"java.lang"*. This package is special in that it is automatically imported. You can still type this package in an import statement, but you don't have to. So, **Option B** is the correct answer. *"java.util"* package exists in every Java class but you have to import this package. "system.lang" and "java.system" do not exist in a standart java class. **Option A**, **Option C** and **Option D** are incorrect.

Q8-) Option C

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 A** 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 D** are examples of multiple-line comment. **Option C** is not a valid type of comment in Java, it is the answer.

Q9-) Option D

A *".java"* file can contain any number of class declaration but only one of them may be a public class. Yet, it don't have to be a public class. According to this information; Option A, **Option B** and **Option C** are incorrect. **Option D** is the correct answer.

Q10-) Option B

As mentioned **Q1**, a main method contains access modifiers *“public”* and *“static”*, return type of *“void”* and a single *“String”* argument. To call main method without creating an object it should be static so, main() method in the example should be corrected.

Static variables are accessible from all classes and methods in the same package because it is default.

Instance variables can be accessible from other classes or static methods by creating object. They are accessible directly from non-static methods which are in their own class.

Private Static variables can not be accessible from other classes directly, we only can access in it’s own class. We can access from other classes by creating an object.

Local variables are not accessible from outside of the method.

The main method here is not static so, from P1 we can access both class and instance variables. Moreover, there is a local variable in scope of main method and we can access it. Namely 2 class variables, 2 instance variables and 1 local variable are accessible from P1. Answer is **Option B.**

Q11-) Option B

Option A and **Option C** is incorrect because, a class is compiled even if it contains unused, unnecessary, or duplicated import statements. If a class contains an import statement for a class used in the program that cannot be found, it can't compile. So, the statement in Option D is false. Removing an unused import statement does not affect compilation of the program. We can remove it and our program can still compile. The answer is **Option B**.

Q12-) Option A

Static variables are accessible from all classes and methods in the same package because it is default.

Instance variables can be accessible from other classes or static methods by creating object. They are accessible directly from non-static methods which are in their own class.

The *"birds"* variable is not static and it requires a class instance variable to access. The main method is static and it cannot access to any class instance variables. This code, does not compile because of the 5th line. **Option A** is the correct answer. To compile the code, we can define the birds variable static. The output becomes "15".

Q13-) Option C

The "***javac"* command** is used to compile Java programs, it takes a .java file as input and produces bytecode. 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"

According to information above, statement "I" and "III" are false. And we also know, Java is an OOP Language. So, statement "II" is false too. None of this statements is true, the answer is **Option D**.

Q14-) Option D

Element	Example	Required?	Where does it go?
Package declaration	`package abc;`	No	First line in the file
Import statements	`import java.util.*;`	No	Immediately after the package
Class declaration	`public class C`	Yes	Immediately after the import
Field declarations	`int value;`	No	Anywhere inside a class
Method declarations	`void method()`	No	Anywhere inside a class

This table demonstrates elements of a class.

Option A: *"import widget.*;"*

This is an import statement and a class definition can start with an import statement if there is no package statement.

Option B: *// Widget Manager*

This is a single-line comment and a comment can be the first line of a Java class file.

Option C: *package sprockets;*

This is a package statement, it should be the first line of a Java class file.

Option D: *int facilityNumber;*

This is a variable definition. A variable definition cannot be the first line of a Java class file. The answer is Option D.

Q15-) Option C

In Java, every class is not required to include a package declaration. Classes may be defined without a package so, **Option A** is incorrect. **Option B** is also incorrect. To create a package, you choose a name for the package and put a package statement with that name at the top of every source file that you want to include in the package. The package statement must be the first line in the source file. It is possible to restrict access to objects and methods within a package. There can be only one package statement in each source file. "*package-private"* (often just called package) means that other members of the same package have access to the item. Namely, **Option D** is incorrect but **Option C** is correct.

Q16-) Option B

As we mentioned at Q13;

"> javac sample.java" is the syntax of compilation command. This command requires ".java" extension.

"> java sample" is the syntax of execution command. This command takes byte code as input without a file extension.

Option B compensates these requirements, it is the answer.

Q17-) Option B

Structuring a Java class such that only methods within the class can access its instance variables is referred to as encapsulation. The answer is **Option B.**

The meaning of encapsulation, is to make sure that "sensitive" data is hidden from users. To achieve this, you must:

- declare class variables/attributes as *"private"*
- provide *"public" "get"* and *"set"* methods to access and update the value of a private variable

Q18-) Option D

Answer is **Option D. The code does not compile** because, the *"height"* variable declared within the if then statement block and we cannot access it from outside the if-then statement. To compile this code, we can define a local *"height"* variable. After this we can access it from outside the if-then statement and get "56" as output.

Q19-) Option A

A Java bytecode file is a file (with the .class filename extension) containing Java bytecode that can be executed on a compatible Java Virtual Machine (JVM). A Java class file is produced by a Java compiler from Java programming language source files (.java files) containing Java classes. If a source file has more than one class, each class is compiled into a separate class file.

JVMs are available for many platforms, and a class file compiled on one platform will execute on a JVM of another platform. This makes Java applications platform-independent so, **Option B** is incorrect.

Java bytecode files contains binary data and binary data is not human readable. It is difficult to read, not easy. **Option C** is not a correct proposition.

After we compile a ".java" code to bytecode, we can execute it without the source file. It does not require the corresponding ".java" that created it to execute. **Option D** is also incorrect.

Q20-) Option A

A **semicolon (;)** is the correct character for terminating a statement in Java.

Q21-) Option C

This code compiles without issue and we get “31” as output.

Q22-) Option C

On line 1, the class keyword is missing. On line 2, there are two types for the variable “count”, this is an invalid declaration. On line 4, there is a “private” access modifier after the return type. Java do not allow this. Line 3 is the only line that does not contain a compilation error. The answer is **Option C**.

Q23-) Option D

JVMs are available for many platforms, and a class file compiled on one platform will execute on a JVM of another platform. Java is a platform independent programming language. **Platform independence** allows a Java class to be run on a wide variety of computers and devices.

Q24-) Option A

As mentioned at Q19 & Q23, JVM translates Java instructions to machine instructions and support the platform independence. JVM also manages memory for the application. Namely, **options B, C and D** are correct. Java bytecode can be easily decoded/decompiled and JVM cannot prevent it from this. **Option A** is not a correct statement, it is the answer.

Q25-) Option B

A local variable is a variable defined within a method. Local variables must be initialized before use. They do not have a default value and contain garbage data until initialized. Variables that are not local variables are known as instance variables or class variables. Instance variables are also called fields. Class variables are shared across multiple objects. And there is no such thing as package variables, so Option A is incorrect.

Local variables -> in scope from declaration to end of block
Class variables -> in scope from declaration until program ends
Instance variables -> in scope from declaration until object garbage collected

Class variables are always in scope for the entire program.

Q26-) Option C

Option A: `television.actor.recurring.Marie`

- The sub-package recurring not imported. This statement is false.

Option B: `movie.directors.John`

- This statement uses "directors" but "director" used in the import statments.

Option C: `television.actor.Package`

- This is a valid class that accessible from the wildcard import and it is the correct answer.

Option D: `movie.NewRelease`

- This is a false statement because the wildcard is applied to *"movie.director"* not to *"movie"*.

Q27-) Option D

Element	Example	Required?	Where does it go?
Package declaration	`package abc;`	No	First line in the file
Import statements	`import java.util.*;`	No	Immediately after the package
Class declaration	`public class C`	Yes	Immediately after the import
Field declarations	`int value;`	No	Anywhere inside a class
Method declarations	`void method()`	No	Anywhere inside a class

According to this table the correct order of statements for a Java class file is; package statement, import statements, class declaration. The answer is **Option D**.

Q28-) Option D

"java.lang" package is special in that it is automatically imported. You can still type this package in an import statement, but you don't have to so, the import statements "java.lang.*" and "java.lang.Object" can be safely remove. Besides, one of "stars.*" and "stars.Blackhole" import statements is redundant. Namely, we can safely remove 3 import statements here and the answer is **Option D**.

Q29-) Option D

The code prints the third argument of *"theInput"* method. To print *"White-tailed"* the third argument of the command must be *"White-tailed"* so, **Option C** is the answer.

Q30-) Option B

As mentioned at Q13 the ***"javac"* command** is used to compile Java programs, it takes ".java" file as input and produces bytecode with ".class" file extension. **Option B** is the correct answer.

Q31-) Option B

Java is not a procedural programming language, it is an object oriented programming language so, **option A** is incorrect. Java doesn't supports "*operator overloading"* this makes **Option C** is false. Java program runs in JVM that controls the memory management. This means that program cannot directly access memory for security and stability reasons so, **Option D** is also incorrect. *"Method Overloading"* is a feature that allows a class to have more than one method having the same name, if their argument lists are different and Java allows this. **Option B** is correct.

Q32-) Option D

Option A: `import, class, null`

This statement is incorrect because *"getMaxWithdrawal()"* has a return statement and "*null"* cannot be return type of a method.

Option B: `import, interface, void`

This statement is also incorrect because *"getMaxWithdrawal()"* has a return statement and *"void"* cannot be return type of a method.

Option C: `package, int, int`

This statement is incorrect because *"Banker"* is a class and to declare a class, *"class"* keyword should be used, not "int" that is a return type.

Option D: `package, class, long`

This is the correct answer. All this keywords are proper for the blank lines. With these keywords the code compile without an issue.

Q33-) Option A

The code compiles without an issue. The input value of "x" at the constructor is ignored and assignment of "end" to be "4". While "end" is "4" and "start" is "2", the result of subtraction is "2". Due to "distance" is 5, we get "2 5" as the output. The answer is **Option A**.

Q34-) Option D

When we use the inheritance, method signature changes in parent classes may break subclasses and program uses extra time/resources at runtime that use overloaded methods but these are arguments against inheritance not the reason of usage of inheritance. **Options B and C** are incorrect. Inheritance does not require that the superclass and its subclasses must be in same package so, **Option A** is incorrect too.
Inheritance supports the concept of "reusability", when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class. Namely, developers minimize duplicate code in new classes by sharing code in a common parent class. **Option D** is correct.

Q35-) Option A

As mentioned at Q8, there are three types of comments in Java. **Option A is a single line comment** example and it is valid.

Q36-) Option B

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 B compensates these requirements and it is the correct answer.

Q37-) Option B

The line of code cannot be inserted at the place of line a1 because, a1 is at outside of the class so, **options A and D** are incorrect.

Line a3 is in a method, in other words it is in local area. We can declare only a local variable in a local area and local variables cannot have access modifiers like "public". This makes **Option C** is incorrect too.

Lines "a2" and "a4" are in a class. We can declare instance variables such as "`public String color`" there so, the answer is **Option B**.

Q38-) Option A

A class definition is the only requirement in a valid Java class file. This makes **Option A** the correct answer. A package statement and import statements are optional while declaring a class, not mandatory. Besides, a class does not have to be public. It can be private or protected also. Namely, **options B, C and D** are incorrect.

Q39-) Option B

In Java, ***".class"*** is the proper file extension for a Java bytecode compiled file. The correct answer is **Option B.**

Q40-) Option C

As mentioned at Q7, *"java.lang"* is automatically imported in all Java classes. Hence, both *"java.lang.Math"* and *"pocket.complex.Math"* are imported into this class but this is not to cause a compilation error so, the **Option A** is incorrect. Line 3 is an unnecessary import but a class is compiled even if it contains unnecessary import statements so, **Option B** is incorrect. Line 6 does not compile due to the class reference is ambiguous. **Option D** is incorrect too and **Option C** is the answer. To use the floor method without an issue and compile the code we have to declare package name of the Math class.

Q41-) Option A

To access a class automaticly without using its full package name, we have to import its package. Remember that *"java.lang"* package is imported by default in Java. In this case, "dog" and *"dog.puppy"* are also imported so, we can automatically access *"dog.Webby"* and *"dog.puppy.Georgette"*. Namely, **options B, C, D** are true statements. But we cannot access *"dog.puppy.female.KC"* because import statements does not include subpackages. The answer is **Option A**.

Q42-) Option A

Option A :Encapsulation is the technique of structuring programming data as a unit consisting of attributes, with actions defined on the unit.

Option B: Object orientation works through the creation, utilization and manipulation of reusable objects to perform a specific task, process or objective.

Option C: Platform independence means that when you compiled a Java code you can execute the program on any operating system.

Option D: Polymorphism is the characteristic of being able to assign a different meaning or usage to something in different contexts specifically, to allow an entity such as a variable, a function, or an object to have more than one form.

Q43-) Option A

"Apple" class is in "*food.fruit", "*Broccoli" class is in "*food.vegetables"* and "Date" class is in *java.util* packages. All these classes are used in "Grossery" class so, all of the import statements in this class are required and if we remove any of them the code does not compile. The answer is **Option A.**

Q44-) Option C

"numLock" variable is *"private" and it is* not accessible in the *"main()"* method without an instance of the *"Keyboard"* class. Because of this, the code does not compile so, the answer is **Option C**. To access *"numLock"* and compile this code, we can create an object of this class.

Q45-) Option D

We should access static variable *"wheels"* by static way but yet, compiler can read this variable here. The code compiles and we get "20" with addition of 4 (feet), 15 (tracks) and 1 (wheels) as the output. **Option D** is the answer.

Q46-) Option B

The code compiles and "*printColor()"* method prints "purple". The value of color passed to the *"printColor()"* method has no importance here. *"printColor()"* method always prints purple as output. The answer is **Option B**.

Q47-) Option C

The *"**javac**"* **command** is used to compile Java programs, it takes a ".java" file as input and produces bytecode with ".class" file extension. This makes statement II is true

The *"**java**"* command uses a period (.) to separate packages, not a slash (/) so, statement I is true and statement III is false. The answer is **Option C**.

Q48-) Option D

The code compiles without issue but does not execute. We encounter a run-time error. Because, the *"main()"* method does not have the correct method signature that mentioned at Q1, "*String"* array parameter is missing. The answer is **Option D**. We need to declare the required input argument which is an array of "String" to compile this code.

Q49-) Option C:

The class diagram shows a class named *"Book"* that has two attributes, one variable (`numberOfPages`) and one method (`getRating()`).

Option A: `public class Book { public int numOfPages;`

This implementation does not compile because, the closing bracket for the class is missing.

Option B:
```
public class Book {
public String getRating() {return null;} }
```

This in not the correct implementation because, "`numberOfPages`" variable is missing.

Option C:
```
public class Book {
public int numberOfPages;
public String getRating() {return null;} }
```

This is the correct implementation. *"Book class"* has two attributes, one variable (`numberOfPages`) and one method (`getRating()`).

Option D:
```
public class Book {
void numberOfPages; }
```

This implementation does not compile because, *"void"* is not a valid type for a variable. Besides, `getRating()` method is missing here.

Q50-) Option C

Garbage collection can happen at any time while the application is running. It is instantaneous and not predictable. JVM cannot predict the schedule so, **Option A** is incorrect. **Option D** is incorrect too because, the computer must be able to run the JVM in order to execute a Java class. **Option C** is the answer because, the JVM does require properly defined an entry point method to execute the application. Yet, I do not know why the statement at **Option B** is incorrect.