Emre KARAGÖZ

ASSIGNMENT 6
JAVA METHODS AND ENCAPSULATION
(HW 6)

"strictfp" keyword in java

strictfp is a keyword in Java used for restricting floating-point calculations and ensuring same result on every platform while performing operations in the floating-point variable. Floating point calculations are platform dependent i.e. different output(floating-point values) is achieved when a class file is run on different platforms (16/32/64 bit processors). To solve this types of issue, strictfp keyword was introduced in JDK 1.2 version by following **IEEE 754** standards for floating-point calculations.

Important points:

- strictfp modifier is used with classes, interfaces and methods only.

```
strictfp class Test

{

    // all concrete methods here are

    // implicitly strictfp.

}

strictfp interface Test

{

    // all  methods here becomes implicitly

    // strictfp when used during inheritance.

}

class Car

{

    // strictfp applied on a concrete method

    strictfp void calculateSpeed(){}

}
```

- When a class or an interface is declared with strictfp modifier, then all methods declared in the class/interface, and all nested types declared in the class, are implicitly strictfp.

- strictfp **cannot** be used with abstract methods. However, it can be used with abstract classes/interfaces.

- Since methods of an interface are implicitly abstract, strictfp **cannot** be used with any method inside an interface.

```
strictfp interface Test

{

    double sum();

    strictfp double mul(); // compile-time error here

}
```

```
/Java program to illustrate strictfp modifier

public class Test
{
    // calculating sum using strictfp modifier
    public strictfp double sum()
    {
        double num1 = 10e+10;

        double num2 = 6e+08;

        return (num1+num2);

    }

    public static strictfp void main(String[] args)
    {
        Test t = new Test();

        System.out.println(t.sum());
    }
}
```

Output:

```
1.006E1
```

**https://www.geeksforgeeks.org/strictfp-keyword-java/

Q1-) Option C

Java offers four choices of access modifier:

- ***public*** - can be called from any class.
- ***private*** - can only be called from within the same class.
- ***protected*** - can only be called from classes in the same package or subclasses.
- ***Default (Package Private) Access*** - can only be called from classes in the same package. This one is tricky because there is no keyword for default access. You simply omit the access modifier.

So, the access modifier ***protected*** allows access to everything the access modifier ***package-private*** does and more. **Option C** is the answer.

Q2-) Option B

In Java, the first statement of every constructor is either a call to another constructor within the class, using ***this(),*** or a call to a constructor in the direct parent class, using ***super().*** **Option B** is the answer.

Q3-) Option D

The code does not compile because the "`sell()`" method does not return a value if the both conditional statements are false. An "else" statement is needed to prevent the code from this case. **Option D** is the answer.

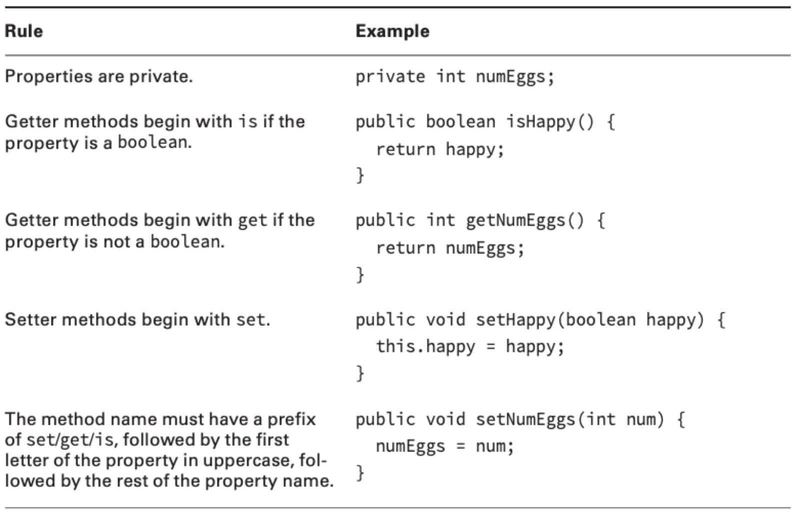
Q4-) Option D

The code does not compile because the "nested()" method which called by "main" does not return a value. "void" type is not allowed there. **Option D** is the answer.

Q5-) Option B

Java is a “pass-by-value” language. This means that a copy of the variable is made and the method receives that copy. Assignments made in the method do not affect the caller. **Option B** is the answer.

Q6-) Option C



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; }`

This table shows rules for JavaBeans naming conventions.

Option A: `public void getArrow()`

A getter method should return a value.

Option B: `public void setBow()`

A setter should take a value.

Option C: `public void setRange(int range)`

This is a consummate setter decleration. It takes a value, does not return a value and the method name is starts with “set”.

Option D: `public String addTarget(String target)`

A setter should not return a value and it should start with “set”.

Q7-) Option B

Option B is the answer because "super()" and "this()" cannot be used together in the same constructor. If "this()" is used, it must be the first line of the constructor.

Q8-) -

```
package ai;
public class Robot {
     compute() { return 10; }
}
```

Option A: `Public int;` "Public" is not a valid access modifier
Option B: `Long;` "Long" is not a valid access modifier
Option C: `void;` Cannot return a value from a method with void result type
Option D: `private String;` requires a "String" but it returns an "int"

All these options cause an error and prevents the code from compiling.

Q9-) Option C

A static variable is common to all the instances (or objects) of the class because it is a class level variable. In other words you can say that only a single copy of static variable is created and shared among all the instances of the class. **Option C** is the answer.

Q10-) Option A

Adding "this(4)" is causes to print "5". Because, "protected boolean outside" defaults to false so, "rope+1" is evaluated in "public Jump(int rope)" function. **Option A** is the answer.

- `rope=4;` ***prints 4***
- `new Q10(4);` ***prints 1***
- `this(5);` ***prints 6***

Q11-) Option B

An instance of one class may not access package-private attributes in a parent class, provided the parent class is not in the same package. Because, they must be in the same package so, **Option B** is false, and it is the answer.

- An instance of one class may access an instance of another class's attributes if it has a reference to the instance and the attributes are declared public.
- Two instances of the same class may access each other's private attributes.
- An instance of one class may access an instance of another class's attributes if both classes are located in the same package and marked protected.

These three statements are all true.

Q12-) Option D

Variable "stuff" is a public variable so, it can be accessed and modified from any class. Filling other two blanks with correct words does not ensure the class data is properly encapsulated **Option D** is the answer.

Q13-) Option C

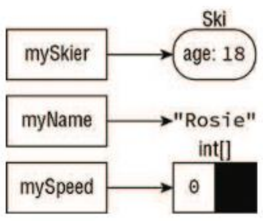
The Java compiler will not always insert a default no-argument constructor if you do not define a no-argument constructor in your class. It only inserts a constructor if there is not any constructors in the class. So, **Option A** is not true. In order for a class to call super() in one of its constructors, its parent class must not explicitly implement a no-argument constructor because the parent can have a default no-argument constructor inserted by the compiler. This makes **Option B** false. **Option D** is also incorrect because, a class may not contain more than one no-argument constructor. This causes an error and prevets the code from compiling. Since If a class extends another class that has only one constructor that takes a value, then the child class must explicitly declare at least one constructor, **Option C** is the answer.

Q14-) Option A

Option A: `public void sing(String key, String... harmonies)`
Option B: `public void sing(int note, String... sound, int music)`
Option C: `public void sing(String... keys, String... pitches)`
Option D: `public void sing(String... notes, String melodies)`

If a method takes a varargs parameter, varargs must be the last element of the argument list. Only **Option A** follows this rule and it is the answer.

Q15-) Option C



As explained in Q5 Java is a ***"pass-by-value"*** language. This means that a copy of the variable is made and the method receives that copy. Assignments made in the method do not affect the caller. Callint to the "slalom" method does not affect to `"mySkier"`, `"mySpeed"` and `"myName"` variables. `"mySkier"` points out a Ski object that has age variable which equals to 18. `"mySpeed"` is "0" because it defaults to "0" automatically and `"myName"` is "Rosie" which is assigned to it. **Option C** is the representation of these variables.

Q16-) Option B

```
public class Calculations {
       public Integer findAverage(int sum) {
              return sum;
       }
}
```

Overloading allows different numbers of parameters. Everything other than the method signature can vary for overloaded methods. This means there can be different access modifiers, specifiers (like static), return types, and exception lists. These are all valid overloaded methods:

- `public void fly(int numMiles) { }`
- `public void fly(short numFeet) { }`
- `public boolean fly() { return false; }`
- `void fly(int numMiles, short numFeet) { }`
- `public void fly(short numFeet, int numMiles) throws Exception { }`

Option A: `public Long findAverage(int sum)`
Option B: `public Long findAverage(int sum, int divisor)`
Option C: `public Integer average(int sum)`
Option D: `private void findAverage(int sum)`

Options A and D are prevents the code from compiling because their parameters and names are the same with method "findAverage". Option C is a different method with a different name but Option B is an overloaded version of "findAverage" and it is the answer.

Q17-) Option D

Encapsulation prevents users from modifying the internal attributes of a class. By doing this, it helps to maintain class data integrity ıf data elements. For encapsulation, remember that data (an instance variable) is private and getters/setters are public. Java defines a naming convention that is used in *JavaBeans*. JavaBeans are reusable software components. So, encapsulation promotes usability by other developers. Namely, **options A, B and C** are true statements about encapsulation but increasing concurrency and improving performance is not certain by implementing encapsulation. **Option D** is the answer.

Q18-) Option A

In Java, arguments are passed by value. When the method modifies an argument variable, then only the copy of the original value is modified. Any primitive value is copied during the method call. When an object is passed as an argument, then the copy of the reference to the object is passed. That way, the object is available to be modified for the method. In the case of classes that have their primitive counterpart, and also in the case of String and some other class types, the objects simply do not provide methods or fields to modify the state. This is important for the integrity of the language, and to not get into trouble when objects and primitive values automatically get converted.
In other cases, when the object is modifiable, the method can effectively work on the very object it was passed to. This is also the way the sort method in our example works on the array. The same array, which is also an object itself, is modified.
This argument passing is much simpler than it is in other languages. Other languages let the developer mix the pass by reference and the pass by value argument passing. In Java, when you use a variable by itself as an expression to pass a parameter to a method, you can be sure that the variable itself is never modified. The object it refers to, however, in case it is mutable, may be modified.
An object is mutable if it can be modified, altering the value of some of its field directly or via some method call. When a class is designed in a way that there is no normal way to modify the state of the object after the creation of the object, the object is immutable. The classes Byte, Short, Integer, Long, Float, Double, Boolean, Character, as well as String, are designed in the JDK so that the objects are immutable. It is possible to overcome the limitation of immutability implementation of certain classes using reflection, but doing that is hacking and not professional coding. Doing that can be done for one single purpose—getting a better knowledge and understanding of the inner workings of some Java classes, but nothing else.

"String", "long" and "boolean" can be modified after they are passed to a method as an argument but an "int[]" can be. **Option A** is the answer.

Q19-) Option B

```
package useful;
public class MathHelper {
     public static int roundValue(double d) {
     // Implementation omitted
     }
}
```

Option A: `MathHelper:roundValue(5.92)`

- This is not a valid syntax.

Option B: `MathHelper.roundValue(3.1)`

- This is valid and it's the best way to call the following method from another class in the same package.

Option C: `roundValue(4.1)`

- This call cannot be used without static import.

Option D: `useful.MathHelper.roundValue(65.3)`

- This is a valid call but writing tha package name ("useful") is redundant because classes are already in the same package.

Q20-) Option D

If the return type is void, the return statement is optional. **Option D** is the answer.

Q21-) Option C

Java does not allow us to modify final variables. To modify "score" and "result" variables and compile this code, we should remove two final keywords before these variables. After this, the code compiles but it throws ArrayIndexOutOfBoundsException. **Option C** is the answer.

Q22-) Option D

"super()" is used to call a constructor in the parent class, ***"super"*** is used to reference a member of the parent class.

Q23-) Option B

`void run(String government)`
There is no keyword for access modifier so, this method is package-private and it can only be called from classes in the same package. **Option B** is the answer.

Q24-) Option A

- Change the access modifier of strength to private.
- Add a getter method for material.
- Add a setter method for material.

Changing the access modifier protected to private improves the encapsulation because this makes "strenght" only be accesible from within the same class. Adding a getter or setter for "material" does not help to encapsulate the data in the class. It makes this variable accessible. Since only the first statement is true, **Option A** is the answer.

Q25-) Option A

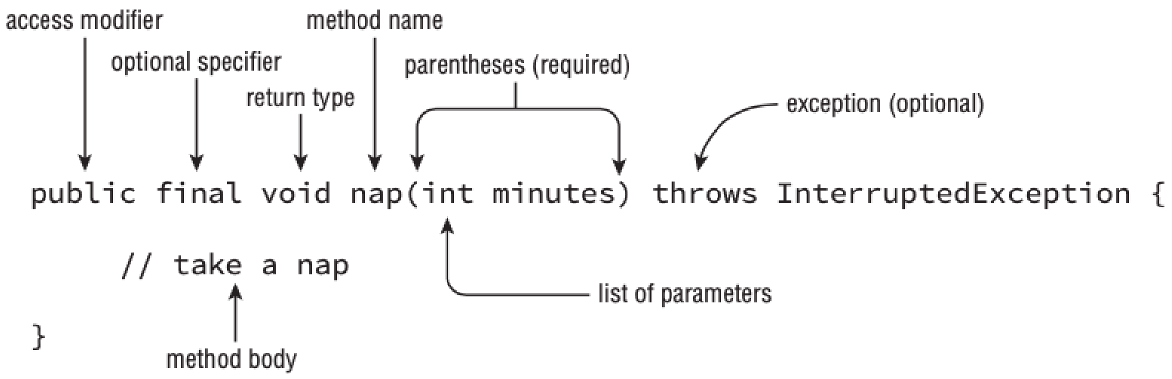
A method name may only contain letters, numbers, $, or _. Also, the first character is not allowed to be a number, and reserved words are not allowed. By convention, methods begin with a lowercase letter but are not required to.
Option A: `Go_$Outside$2() //valid`
Option B: `have-Fun() // "-" is not allowed`
Option C: `new() // "new" is  reserved word`
Option D: `9enjoyTheWeather()// first character cannot be a number`

Only **Option A** follows these rules and it is the answer.

Q26-) Option D

```
package farm;
public class Coop {
     public final int static getNumberOfChickens() {
     // INSERT CODE HERE
     }
}
```

Option D is the answer. The code does not compile no matter which line we add because, this method signature is not valid. The return type “int” should be written after final, static keywords and before the name. A valid method signature must be as follows:



Q27-) Option B

In Java, arguments are passed by value. When the method modifies an argument variable, then only the copy of the original value is modified and a change made to a primitive value passed to a method is not reflected in the calling method. So, options A and D are incorrect. Reassigning an object reference passed to a method is not reflected in the calling method, because they are different objects from each other. This makes Option C incorrect too. A change made to the data within an object passed to a method is reflected in the calling method because the object that the copied references to the same point. **Option B** is the answer.

Q28-) Option C

The code does not compile due to the final keyword. The “contents” variable marked final and we cannot assign a value to final variable “contents”. If we remove “final” from declaration, Option A would be a possible output but **Option C** is the answer of this question.

Q29-) Option A

As explained the table in Q6, JavaBeans uses ***"get"*** and ***"set"*** prefixes for getter and setter methods and ***"is"*** prefix for boolean's getter methods. **Option A** is the answer.

Q30-) Option C

To call "getClothes()" without the its class name "Store", we have to implement a static import. **Option C** (`import static clothes.Store.getClothes;`) is the answer, it has the correct syntax because static imports are used with members of the class. Option A would be the answer if "getClothes()" were called with its class like this: "`Store.getClothes()`".

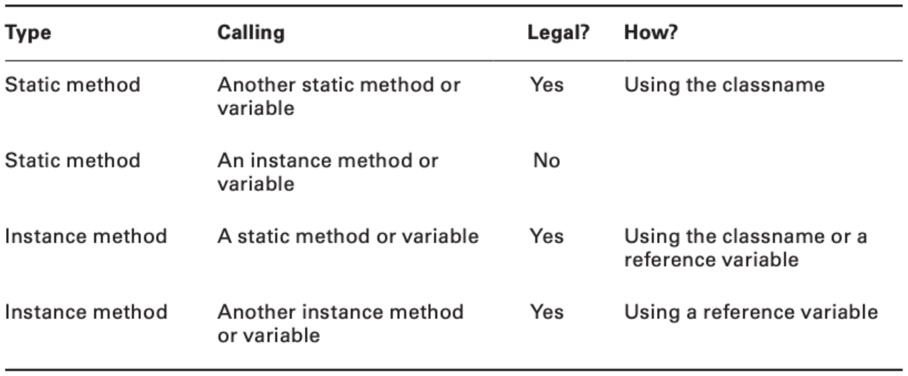
Q31-) Option D

Default (Package Private) Access: can only be called from classes in the same package. This one is tricky because there is no keyword for default access. **Option D** is the answer.

Q32-) Option B

There is only one line to prevent this code from compiling. "`public void Stars()`" is not a constructor method because constructor methods do not have return types. Since it is not a constructor, "super()" cannot be called by this method. Only constructors can call "super()". **Option B** is the answer.

Q33-) Option A



Type	Calling	Legal?	How?
Static method	Another static method or variable	Yes	Using the classname
Static method	An instance method or variable	No	
Instance method	A static method or variable	Yes	Using the classname or a reference variable
Instance method	Another instance method or variable	Yes	Using a reference variable

A static method or initialization block is not allowed to reference an instance variable. This makes Option C and B false statements. Option D says us a final static variable may be set in a constructor, but it is not true. We have to initialize static variables when declare them. **Option A** is the answer because as we can see from the table above, an instance method is allowed to reference a static variable.

Q34-) Option B

calculateDistance() requires “short” or a return type that can be converted to short as return type. Options A and C are incorret because “Integer” and “Long” are larger data types than short, this data types couses compilation error. Since, we can easily promote a “byte” to “short. **Option B** is the answer. Option D (`return new Short(4).longValue()`) is also incorrect and couses an error because compiler assumes “4” is an integer.

Q35-) Option C

Overloading allows different numbers of parameters. Everything other than the method signature can vary for overloaded methods. This means there can be different access modifiers, specifiers (like static), return types, and exception lists. According to this;

- Overloaded methods must have the same name.
- Overloaded methods must have the same return type.
- Overloaded methods must have a different list of parameters.

The first and third statements about overloaded methods are true but the second statement is false because overloaded methods can have the same return types. **Option C** is the answer.

Q36-) Option B

```
package work;
public class Week {
     private static final String monday;
     String tuesday;
     final static wednesday = 3;
     final protected int thursday = 4;
}
```

- `private static final String monday;`

  this line causes an error because a value must be assigned to static variable `“monday”`

- `final static wednesday = 3;`

  this line causes an error because return type is missing

These two lines need to be removed for the class to compile. **Option B** is the answer.

Q37-) Option D

“public void Puppy” is not a contructor so, “new Puppy(2).wag” (g3) prevents this code from compiling. Constructor methods do not have return types. **Option D** is the answer.

Q38-) Option A

The ***public*** access modifier allows access to everything the ***private*** access modifier does and more. "private” access modifier only allows access to members of the same class but public access modifier allows access to members of any class.

Q39-) Option A

Java is a “pass-by-value” language. This means that a copy of the variable is made and the method receives that copy. Changes in the “sendHome()” method does not affect the object “phone” that was passed in. “phone.size” remains being “3” after passed to method. The code compiles without an issue and outputs “3”. **Option A** is the answer.

Q40-) Option B

```
public class Drink {
     public static void water() {}
     public void get() {
          // INSERT CODE HERE
     }
}
```

- `water();`
- `this.water();`
- `Drink.water();`

We can call method water() by using these three lines. Code compiles without an issue with these lines but “`this.Drink.water();`” causes an error an prevent this code from compiling.

Q41-) Option C

```
public void call(int count, String me, String... data)
```

This method requires to an int, a String, and String varargs respectively. **Option C** (`call(2,"home","sweet")`) follows this and it is the answer.

Q42-) Option D

Option A: The value of a static variable must be set when the variable is declared or in a static initialization block.

- This statement is true about only final static variables

Option B: It is not possible to read static final variables outside the class in which they are defined.

- This statement is not true because static final variables are readable from outside the class if they are not ***"private".***

Option C: It is not possible to reference static methods using static imports.

- This statement is not true because static imports can be used to reference static methods and static variables.

Option D: A static variable is always available in all instances of the class.

- This is a correct statement, a static variable is accessible from all instances of the class.

Q43-) Option A

Option B is true because a class does not have to have a constructor explicitly defined. The Java compiler inserts a default no-argument constructor if you do not define a no-argument constructor in your class. A constructor may pass arguments to the parent constructor so, **Option C** is also true. A final instance variable whose value is not set when they are declared or in an initialization block should be set by the constructor and this makes **Option D** is true. Option A tells us "the first line of every constructor is a call to the parent constructor via the super() command." but this is not true. In Java, the first statement of every constructor is either a call to another constructor within the class, using ***this(),*** or a call to a constructor in the direct parent class, using ***super().*** **Option A** is the answer.

Q44-) Option D

The code will not compile regardless of the number of final modifiers removed, **Option D** is the answer. Variable "height" is an instance variable, not a static. So, it cannot be referenced from a static initialization block.

Q45-) Option D

The compiler inserts no-argument a super() call to RainForest() because a constructor call is not the first line of the RainForest() constructor. But, this couses a compilation error because class "Forest" which is the RainsForest's parent class does not have a no-argument constructor. **Option D** is the answer. To compile this code we should insert a no-argument constructor to class "Forest".

Q46-) Option A

The evaluation order of Java is from left to right.
`"choose((byte)2+1)"`
Firstly, "2" is cast to byte, then after the addition an "int" value is passed to the method. Because, "+" operator automatically promotes byte to int. The code compiles and prints "5".

Q47-) Option C

The code does not compile because method "getScore" requires to "Long" but in the main method, an "int" value is passed to this method. "int" is a primitive. It cannot be converted to Long. **Option C** is the answer.

Q48-) Option A

As explained in Q25, 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: `$sprint()     // follows the rule before`
Option B: `\jog13()      //invalid first character`
Option C: `walk#()       //# is not allowed in a method name`
Option D: `%run()        //invalid first character`

Q49-) Option B

"protected" access mofier allows any subclass of Bouncer or any class in the same package as Bouncer to access this variable. **Option B** is the answer.

Q50-) Option D

None of these imports allows the second class “Teller” to compile. To compile it, this class can access methods “withdrawal()” and “deposit()”. Static imports provides access to static methods, but these methods are not static methods. An instance of class “Bank” is required to access these methods. **Option D** is the answer.