PayCore Java Spring Bootcamp [2.3]

Homework 1# 09.01.2022

Question 1: Why we need to use OOP? Some major OOP languages?

Answer:

First of all, lets define the Object Oriented Programming. We can say that Object Oriented Programming is the way we are defining our everything that we can see and touch for the computer environment. When we do that we use classes. Classes are things that transferred on computers environment as objects. We use OOP, because;

1. It makes our code more readable.

2. It makes our code easier. Let’s define easier. When we use OOP, we include classes to our codes. With doing this, we are defining every characteristics and methods for our classes and we can use this characteristics and methods wherever we want.

3. It makes our codes reusable. With using classes with OOP logics, we also provide using the same code again and again. That makes our code reusable. For example, let’s say that we are making a car class for our program. We can define car and create a new car object. Than we can use this object with its functions whenever we want.

We can example; C++, Java, C#, Phyton, Delphi, Swift, Ruby for some major Object Oriented Programming Languages.

Question 2: Interface vs Abstract class?

Answer:

Let’s define interfaces first. Interfaces are used to achieve abstraction and multiple [inheritance in Java](https://www.javatpoint.com/inheritance-in-java). Interfaces are not considered as classes. But they can hold references like classes. When we implement classes to Interfaces, we must add operations in interfaces to our classes. We cannot create objects from interfaces. For example, let’s say that we created new interface named ICustomerDal. We cannot use “ICustomerDal icustomerdal = new ICustomerDal();” code to create new interface. But we can create new classes which implemented to our interface. Let’s say that we created new class named MySQLCustomerDal which implemented to ICustomerDal. We can create new object named mysqlcustomerdal with using the code “ICustomerDal mysqlcustomerdal = new MySQLCustomerDal();”. We can do that because we implemented MySQLCustomerDal before. This is also a polymorphism example.

Abstract classes can only inherits one class. Unlike interfaces, abstract classes are considered as classes. When we use abstract class, if we have another classes which inherited from abstract class, we must use abstract class’ methods for these classes. The class have to have an abstract signature to be an abstract class. For example: “public abstract class Animal”. Abstract classes are also used for hiding base class. For example, let’s say that Animal class is the base class and it is an abstract class. And we have 2 classes which are inherited from Animal class, named Cat and Dog. We cannot create a new Animal object from our Animal class. Because our Animal class is an abstract class. With code, we cannot write “Animal animal = new Animal”. Instead we can write “Animal cat = new Cat();. This is also a polymorphism example. We created a new Cat from our Animal class.

Question 3: Why we need equals and hashcode? When to override?

Answer:

We need equals when we compare 2 objects and see if they are the same object or not. But when we use that we need to be careful about our codes. Let’s say we created 2 Bike objects bike1 and bike2. And we give the parameters and definings the same for both of these bikes. If we compare them with code “bike1 ?= bike2” we will get false even we make everything same. Because bike1 and bike2 has different addresses in heap. To see if they are equal we can use codes:

public boolean equals(Object o){

if(this == o) return true;

if(!(o instanceof Bike)) return false;

Bike that = (Bike)o;

return (getColorOfBike() == that.getColorOfBike()) && (getGearValue() == that.getGearValue());

With doing this we are Overriding equals codes and defining 2 objects and comparing them. That will provide us to see if the objects are equal or not.

When we create object, Java will automatically give that object a hash number. But even if we create same 2 objects but we name those objects differently, their hash codes will be different from each other.

Question 4: Diamon problem in Java? How to fix it?

Answer:

Diamon Problem is a problem which is blocking us from doing multiple inheritance in Java. When we use inheritance for a class and define our class’ base class, we cannot give that class another base class. This process is named multiple inheritance and Java is not supporting this process with using Inheritance. To solve this problem we are using Interfaces. Interfaces, allow us to make a class implement from many classes.

Question 5: Why we need Garbage Collector? How does it run?

Answer:

Garbage Collector is a part of Java Memory Management System. It is inside of JVM. And this is one of the most important qualities in Java. In C and C++ when we write a code, we must handle Memory Management System. But in Java, Garbage Collector is handling with this.

Garbage Collector is an Automatic Memory Management Mechanism. Garbage Collector checks heap and tries to identify the unused and unreferanced objects. When Garbage Collector does that it will delete them and clears the memory. Let’s see it with codes:

public class Dog{

String name;

int age;

public Dog(String name, int age){

this.name = name;

this.age = age;

}

}

public static void main(String args[]){

Dog dog1 = new Dog(“Karabas”, 3);

Dog dog2 = dog 1;

System.out.println(dog1.name);

}

Here in this code, we created 2 dog objects but the second dog has the same reference with first dog. When the program is running, our output will be “Karabas” and the Garbage Collector will delete the dog2.

Question 6: Java ‘static’ keyword usage ?

Answer:

For static variables, once they are created, our system will open a place in memory for this static variable. They don’t get another place in memory while we are creating objects every time.

Because static variables are created without being bound to the object, they are accessed without creating the object. Therefore, static variables can be accessed even after no objects are created or all existing objects are deleted.

Our main method is also a static method. When we run the program, main method always runs.

Static keyword makes our methods, variables permanent. This keyword allows us to access these methods, variables without creating their class. And this keyword is important. Because it is directly related with Object Oriented Programming.

Question 7: Immutability means? Where, How and Why to use it?

Answer:

Let’s say that we want to make a class. When we make our class if we can’t change it’s qualities with our operations, that means our class is immutable. Let’s take a look at String class;

String name = “Yılmaz”;

name.toUpperCase();

System.out.print(name);

When we write these code our output still will be “Yılmaz”. That is a proof that String classes are immutable. To change immutable classes we have to create another object. For example;

String name = “Yılmaz”;

name2 = name.toUpperCase();

System.out.print(name2);

This time, our output is YILMAZ. Because we created another string and we printed our second string value. All Wrapper Classes are immutable. For example; Boolean, Short, Integer, Long, Float, Double, Char, String.

We need immutable class, because we sometimes want to make our classes thread safe. When we use immutable classes, because values of our object won’t change, our object will always work on same values. This situation has a disadvantage too. If we want to change values in our object, we must create a new object. This means time and memory waste.

If we want to create an immutable class, we must use final keyword. And we also have to set our class’ accees modifier to private. When we add constructor to our class, we define our classes attributes in that constructor and we cannot change our attributes after that.

Question 8: Composition and Aggregation means and differences?

Answer:

Composition and Aggregation are both means association between classes. Composition is a strong type association. With strong type association we mean, objects which cannot be exist if one of them doesn’t exist. In this association, we have 2 or more objects and 1 of our object has the other ones. We can give a car and an engine example. A car must have an engine. And if an engine doesn’t exist, our car won’t work effectively. And if a car doesn’t exist, our engine won’t work effectively either. Aggregation is a week type association. With week type we mean, objects which can be exist if one of them doesn’t exist. In this association, we also have 2 or more objects and 1 of our object has the other ones. We can give a football team and a football player example. Football team has many football players and if team doesn’t have one of the players, team still exists. And also if player doesn’t have a team, he-she still exists.

Question 9: Cohesion and Coupling means and differences?

Answer:

In a program, there are many classes which is doing multiple tasks. This classes contact each other and work together. In this case, there is a dependency between classes. This dependency increases with how much a class knows about another classes. In projects, these classes must work with each other, so we cannot provide this dependency. In this case, we should make loose coupling.

Coupling is about how much a class knows about other classes and how it accesses class members. Loose Coupling is an approach to defining classes in accordance with a good encapsulation rule and minimizing the reference variables of classes to each other. Tight Coupling is opposite approach against loose coupling. We don’t encapsulate referance variables and classes and, this is making our code worse. We always want to use Loose Coupling approach in our codes.

Cohesion is about how is the class designed. In this approach, each class should have a well defined goal. And also, the responsibility and role of each class should be clearly defined. All classes should specific roles and they must have not much responsibilities.

Question 10: Heap and Stack means and differences?

Answer:

Stack and Heap are both logical part of the RAM. While value types, pointers and addresses are stored in Stack, reference values ​​are stored in Heap. Value types are; int, short, byte, long, decimal, double, float . Reference types are; arrays, String, Classes, Interfaces, Pointers. Stack is working with LIFO(Last in First Out) logic. In Stack, we can delete our data immediately but in heap, Garbage Collector is dealing with deleting process. The data in the stack is placed directly in the memory, so its access is very fast. Heap, on the other hand, is used at runtime, and since it is a scattered memory structure, it is not as easy to access as a stack, so it works slowly.

Question 11: Exception means ? Type of Exceptions?

Answer:

Exceptions are errors when our program is running or at runtime. Some of these errors can be tolarated but some of them can stop our entire program while our program is running. We can catch the exceptions, tolerate them and if it is possible, letting the program works.

In Java, the source of every exception is Throwable class. Throwable class has 2 sub classes. Error and Exception. The Error class is used for JVM related errors that are not directly related to our code and these errors are beyond our control. That is why we cannot control them. But exception is directly connected with our code so we can handle with them in our code.

Type of Exceptions:

1. Checked Exception:

Checked exceptions are also known as compile-time exceptions as these exceptions are checked by the compiler during the compilation process to confirm whether the exception is handled by the programmer or not. If not, then the system displays a compilation error. For example, SQLException, IOException, InvocationTargetException, and ClassNotFoundException.

* 1. SQL Exception:

This type of exception occurs while executing queries on a database related to the SQL syntax.

* 1. IO Exception:

This type of exception occurs while using file I/O stream operations.

* 1. ClassNotFoundException:

This type of exception is thrown when the JVM is not able to find the required class. It may be due to a command-line error, a classpath issue, or a missing class file.

* 1. Invocation Exception:

This type of exception wraps an exception thrown by an invoked method or a constructor. The thrown exception can be accessed with the help of the getTargetException method.

1. Unchecked Exception:

The unchecked exceptions are those exceptions that occur during the execution of the program. Hence they are also referred to as Runtimeexceptions. These exceptions are generally ignored during the compilation process. They are not checked while compiling the program.

1. Null Pointer Exception:

This type of exception occurs when you try to access an object with the help of a reference variable whose current value is null or empty.

1. ArrayIndexOutofBound

This type of exception occurs when you try to access an array with an invalid index value. The value you are providing is either negative or beyond the length of the array.

1. IllegalArgumentException

This type of exception occurs whenever an inappropriate or incorrect argument is passed to a method. For example, if a method is defined with non-empty string as parameters. But you are providing null input strings. Then, the [IllegalArgumentException](https://docs.oracle.com/javase/7/docs/api/java/lang/IllegalArgumentException.html) is thrown to indicate the user that you cannot pass a null input string to the method.

1. IllegalStateException

This type of exception occurs when the state of the environment does not match the operation being executed.

1. NumberFormatException

This type of exception occurs when you pass a string to a method that cannot be converted to a number.

1. ArithmeticException

This type of exception occurs when you perform an incorrect arithmetic operation.

Question 12: How to summarize ‘clean code’ as short as possible?

Answer:

Clean code is really important for programmers. It is based on readability of our code. When we start project and than 6 month later, when we turn back and look at that project, if we cannot read our codes and understand them well, our code is not clean code. Clean code is;

* Easy to understand
* When you look at the codes, you know the function of those codes
* 6 months later, when you turn back, you can still understand what will this code do.
* And when someone looks at that code, they can understand it too.
* When you want to change some stuff with your code, you can easily change it in your clean code.
* It is easy to test and practice too.

Question 13: What is the method of hiding in Java ?

Answer:

To hide methods we use encapsulation. Encapsulation is a method that we use when we don’t want user to get access our variables in our objects directly. We can do it with using access modifier named “private”. And than we can use setter and getter functions to see our variables and set them.

Question 14: What is the difference between abstraction and polymorphism in Java?

Answer:

Polymorphism is known as many forms for classes when we use Inheritance in our codes. Polymorphism allows us to perform a single action in different ways. Polymorphism is one of the foundations of OOP.

Abstraction is used to hide certain details and only show the essential features of the object. Abstraction is one of the foundations of OOP as well.