3. Basic concepts of object-oriented paradigm. Class, object, instantiation. Inheritance, class hierarchy. Polymorphism, method overloading. Scoping, information hiding, accessibility levels. Abstract classes and interfaces. Type members.

***what is a class :***

We group the related variables and methods together, and it becomes a class. It has member values and attributes, and implementations of behavior (member functions and methods)

It can also reflect the real things in the world

***Class and Object :***

Class is a blueprint for creating objects, it describes how objects will look like and how will it behave.

Objects are instances that are created from classes. They may have the values and behaviors that follow the structure of the class.

We can create multiple objects from a class.

***Instantiation :***

Since class is the blueprint for object ,

Instantiation is the process that taking in a class definition and creating an object on it for our program.

***Class hierarchy :***

A class can have subclasses that can inherit all or some of the characteristics of the class. In relation to each subclass, the class becomes the superclass.

Subclasses can also define their own methods and variables that are not part of their superclass.

The structure of a class and its subclasses is called the class hierarchy.

***Encapsulation :***

Encapsulation is the idea of bundling the data , including variables we need and methods that operate on that data, into one single unit.

One good example is class, we encapsulate everything related into a class.

Encapsulation may also refer to a mechanism of restricting the direct access to some components of an object, we can hide variables or the method from outside. 减少代码的耦合性

Information hiding through Encapsulation allows us to change the implementation without changing the expected outcome.

***Benefits of Encapsulation:***

Hiding Data: Users can use the class / method without knowing how the class is implemented. Like a black box.

Easy to Reuse:

You can create different objects depend on the blueprint. Reduce the

With encapsulation it's easy to change and adapt to new requirements.

More Flexibility: Enables you to set variables as red or write-only. Examples include: setName(), setAge() or to set variables as write-only then you only need to omit the get methods like getName(), getAge() etc.

***Abstraction***

Make interface to build object simpler

Help us to reduce the impact of change. Eg : change private methods

***Inheritance :***

Inheritance is the procedure in which one class inherits the attributes and methods of another class.

The mechanism that helps us to reduce redundant code

We don’t have to define everything all over again

***Polymorphism :***

Poly means : many

Morph means : form

Polymorphism means many forms

Polymorphism helps us to reduce long if and else or switch case statements.

It can behave differently depends on different input

it describes the concept that you can access objects of different types through the same interface. Each type can provide its own independent implementation of this interface.

Eg:

We can implement different toString method in different classes , and the method will behave differently depending on the object we are calling.

Types of polymorphism :

Subtype polymorphism (Runtime)

Subtype polymorphism is the most common kind of polymorphism. It is usually the one referenced when someone says, “The object is polymorphic.”

A subtype polymorphism uses one class name to reference multiple kinds of subtypes at once.

Eg:

List<> list = new ArrayList<>() //example of runtime polymorphism

Parametric(static) polymorphism (Overloading)

A parametric polymorphism specifically provides a way to use one function (the same code) to interact with multiple types.

Ad hoc polymorphism (Compile-time)

Java + ???

Coercion polymorphism (Casting)

Coercion polymorphism is the direct transformation of one type into another. It happens when one type gets cast into another type.

Examples of polymorphism :

Method overloading is a static polymorphism

Method overriding is a runtime polymorphism

***Benefits of OOP:***

Encapsulation: Reduce complexity + increase reusability

Abstraction: Reduce complexity + isolate impact of changes

Inheritance: Eliminate redundant code

Polymorphism: Refactor long if /switch cases

***method overloading***

Method overloading is a form of polymorphism in OOP .In method overloading , the methods behave according to their argument types and number of arguments.

Method overloading is an example of Static Polymorphism.

It must have either one of the following form:

1. Number of parameters.

add(int, int)

add(int, int, int)

2. Data type of parameters.

add(int, int)

add(int, float)

3. Sequence of Data type of parameters.

For example:

add(int, float)

add(float, int)

Invalid case of method overloading:

When I say argument list, I am not talking about return type of the method, for example if two methods have same name, same parameters and have different return type, then this is not a valid method overloading example. This will throw compilation error.

int add(int, int)

float add(int, int)

***method overriding***

If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Method overriding is used for runtime polymorphism

### Scoping

<https://www.codecademy.com/article/variable-scope-in-java>

Java Scope

In Java, variables are only accessible inside the region they are created. This is called scope.

***information hiding,***

We don’t need everything in our system to know about everything else. The module should only have access to the data that it needs to do its job. We do this by information hiding.

Information hiding allows modules of our system to give others the minimum amount of data needed to keep work , and hide the others.

It allows programmers to use the module without knowing the implementation details of the module. They can only use the module through interface.

In general, things might change – like implementation details should be hidden, and things are not change should revealed through the interfaces.

Encapsulation is a common technique programmers use to implement information hiding.

We use encapsulation to bundle attributes and behaviors into appropriate class and expose interface to provide an access.

Using access modifiers to implement information hiding.

Information hiding allows us to build flexible, reusable and maintainable systems.

***accessibility levels.***

Access levels are there to ensure the features of an object are called only by those parts of the logic that are allowed to access them. They enforce encapsulation, separation of implementation logic.

Pretty much same with access modifiers??

***Access Modifiers***

Access modifiers set which classes are able to access attributes and behaviors. They also determine which attributes and behaviors that a superclass will share with its subclasses.

In Java, there are four access modifiers that restrict the accessibility of the method or variable to which the modifier is applied. They are only used within classes, not within methods. public and private are the most relevant modifiers to our work, but we will briefly discuss all of them.

private: The most restrictive modifier. It limits access to methods and variables to the class in which they are declared. private is chosen when there is no need to use certain methods or variables outside the class.

We can only access in the encapsulating class itself. Not even the subclasses.

The subclasses cannot override the private method declared in superclass also.

(what about inner class ??)

We can access private members and method in the inner class declared in the encapsulating class.

Inner classes are a security mechanism in Java. We know a class cannot be associated with the access modifier private, but if we have the class as a member of other class, then the inner class can be made private. And this is also used to access the private members of a class.

Creating an inner class is quite simple. You just need to write a class within a class. Unlike a class, an inner class can be private and once you declare an inner class private, it cannot be accessed from an object outside the class.

default: Allows access only from within the current package. If there is no specified access modifier, the method or variable will take on this one.

protected: Allows access to a method or variable only from within the current package, unless it is accessed through a child class outside of the package.

public: The least restrictive modifier. It allows access to a class, method or variable not only from within the class in which it is declared, but outside as well.

Public attributes are allowed to accessed by any class in your system, other classes can retrieve and modify the attribute.

Public methods are also allowed to accessed by any class in your system, but this access does not allow other classes to change the implementation of the behavior. A public method simply allow other classes to call the method and receive any output from it.

Difference between default and protected:

The protected specifier allows access by all code in same package, and all subclasses of the class in, no matter in the same package or not. The default specifier allows access by other code in the same package, but not by code that is in subclasses residing in different packages.

Abstract classes and interfaces.

Abstract classes:

Have at least one abstract method, or is an empty class.

Abstract class can be instantiated.

Interface:

Interfaces cannot be instantiated, it can only have abstract methods.

Type members.