**Classes and Objects**

* Object is an entity that contains both data **(attribute)** and behaviors **(method)**
* Access to attributes and methods **–members--** of an object can be controlled 🡪 data hiding
* Object communication: method call
* The **state** of an object is the data stored within the object – **object attribute**
* The behavior of an object is what the object can do – **object method**
* Each attribute would have corresponding methods such as getter and setter
* Class is a blueprint or template for an object. The class **defines** attributes and methods that all object created within the class will pose. When we **instantiate** an object, we use a class as the basic to build the project.
* Ex: For class Student, there are attributes ID, Name and methods of doExam(), signUpCourse(),… so all of students objects to be created in the future will have all of that attribute and methods.
* For each object instantiated, there is a concrete state that may differs from object of the same class. The class does not usually hold any concrete value
* Access modifier: to restrict the accessibility of your data and method.
* One of the primary advantages of using objects is that it does not reveal all its attributes and behaviors.
* **Instance variable:** is a variable that is declared in a class but not in a constructor or methods. It can be accessed by any member of the class.

**Controlling access**

**Access modifiers:**

* A class’s **public** members are accessible wherever the program has reference to an object that class or one of its subclasses.
* A class’s **private** members are accessible only within the class itself
* A **superclass’s** protected members can be accessed by members of that superclass, by members of its subclasses and by members of other classes in the same package protected members also have package access

**Why private variable:**

* We might lose control of our variables when we work in a team. It prevents data from being accessed and modified accidentally by a class in another part of a program.
* **Constructor**: when an object of a class in created, its instance variables are initialized by default, thus constructor is a used to initialize values upon creating an project. A special method with modifier but no return type. The same name with the class.

**Formal definition:**

* When an object of a class is created, its instance variable is initialized to null by default. A constructor is a special method with the same name as the class which allows value to be initialized to instance variable when the object is created. In fact, it is required for every object created to have a constructor called
* The “new” keyword allows request memory from the system to store an object, then calls the corresponding class’s constructor to initialize the object. The call is indicated by the parentheses after the class name
* By default, the compiler provides a default constructor with no parameters in any class that does not explicitly include a constructor. The instance variables are then initialized to default values
* Explicitly created constructor can be used to specify custom initialization for objects.
* Overloading methods of the same name declared in the same class but different sets of parameters. When an overloaded method is called, the compiler selects the appropriate method by examining the number, types, and order of arguments in the call.
* Static: you can access the methods of the class. You do not have to create an object to use the methods. It allows variable or methods available for all object.

**Inheritance**

* Introduction: When declaring a class, rather than declaring completely new members, we can designate the new class that should inherit the member of an existing class.
* Direct superclass: is the superclass from which the subclass explicitly inherit
* Indirect superclass: is any class above the direct superclass in the class hierarchy
* Single inheritance: a class in derived from exactly 1 direct superclass

**Inheritance issues:**

* A subclass can always be extended from parent class
* A subclass can also inherit the changes to instance variable of the superclass
* Superclass tends to be more general while subclasses tends to be more specific
* The subclass can override (reimplement) the subclass method to have more control
* Each subclass can be a superclass of future subclasses as there are different level of inheritance
* We can declare new things to our class
* The subclass can restrict the superclass data
* A new class created has everything from the given class. However, private data remains private. The subclass has the information but not the access.
* **Protected** members protected only provide access to members of the class
* **@Override** modify the methods that a subclass inherit from its superclass
* **Constructor** the system will call the constructor of the parent class first, either explicitly or implicitly. Meaning, if B extends from A and C from B, all constructor A B C will be called.
* If you have a custom constructor, the implicitly called constructor in subclass will have an error.

**Polymorphism**

* Allows you to write a program that process object that share the same superclass as if they are all object of the same superclass; this can simplify programming
* Ex: Animal respond to a move message, if the “Animal” is a fish it will swim or fly if it is a bird
* 🡪 Help add new features without breaking old code
* You can not call a method of a subclass from the superclass.

**Abstract class**

* The goal is to have 1 method in the superclass and the subclass to no ignore it
* If we declare an abstract class, we can not initiate object out of that class
* The subclass can not be compiled if abstract method from the superclass is not implemented, you can change the subclass to an abstract class when it becomes a concrete class with all methods implemented
* You can abstract superclass to declare variable

**About Abstract class**

* Uses as superclass in inheritance hierarchies
* Can not be use to instantiate objects
* Subclasses must be declare the “missing piece” to become “concrete classes” otherwise the subclasses will also be abstract
* Abstract methods do not provide implementation
* Constructors and static methods can not be declared abstract
* Can use abstract superclass names to involve static methods declared in abstract superclass

**Interface**

* Offers a way for unrelated classes to implement a set of common methods. Interface specifies what operations a radio must permit users to perform but does not specify how the operations are performed. Java interface describes a set of methods that can be called on an object.
* It does not make sense if you have is-a for everything, so you can use interface to link your classes
* Interface is a list of public implemented methods
* Interface may not specify any implementation details such as concrete method declarations and instance variables
* Interface declarations begins with the keyword interface and contains only constants on abstract methods
* All of the interface members by default must be public
* All fields are implicitly public static and final superclass….