OBJECTS AND CLASSES I

OOP

- OOP as a methodology that organizes a program into a collection of interacting objects.
- A more technical definition asserts that OOP is a programming paradigm that incorporates the principles of *encapsulation* , *inheritance* , and *polymorphism* .

- Procedural VS OOP

Objects

- An Object has characteristics or attributes and actions or behaviors.
- E.g. a car, an ATM, a bank account, a deck of cards, a dog, an elephant, a person, a square, a rectangle, a circle, a movie star, a shooting star, a computer mouse, a live mouse, a song, a contact etc.

Class

- A class is a template, or blueprint, from which objects are created.
- As you know, every primitive variable is tied to a data type such as int, char, double, or boolean. Likewise, every object is a member of a *class.*
- As a builder creates houses from the specifications of a blueprint, a program creates *objects* from the specifications of a class. From one blueprint, a builder can build many individual houses; and from one class, a program can create many objects
- A class describes how data and methods are encapsulated as a single object. Every object is an *instance* of some class..

- A class that is not been placed in any package is been placed in java default-package automatically.

Some Java Libraries and Packages

- import is used to import the classes not packages.
- e.g.
  - Java.awt -> abstract window toolkit for graphics programming
  - Java.text -> for text formatting
  - Java.util -> utitlity classes
    - Java.util.Scanner
    - Java.util.zip (package)
    - Java.util.zip.ZipFile
    - Java.util.zip.ZipEntry
    - Java.util.Random

Encapsulation

- Encapsulation is defined as a language feature that comprises the attributes and behaviors into a single entity. That is, data and methods comprise a single entity.
- All objects have there own data, attributes, properties and share same methods, behaviors, actions.

Encapsulation and Information hiding

- Encapsulation: Encapsulation is a technique that bundles together the data and methods into a single unit.
- Information hiding: Information hiding is the principle that hides the implementation of a class.
- In general, information hiding is the principle that hides implementation details from a client class. When implementation details are hidden, all access to the attributes of a class is through its public methods.

Questions

- Why we set data members as private?
  - Because classes are always revised and rewritten. If some data member is public and its value is changed outside of the class, then it must have to be changed in all client classes of this class. But if we set it private then there will be no problem.
- Information hiding allows the classes to be revised without affecting its client.

Example of encapsulation without information hiding

- class public TwoNumbers
- {
- **public** int one; //accessible from outside of class
- **public** int two; //accessible from outside of class
- public MyClass() // default constructor
- {
- one 1;
- two 2;
- }
- public int sum()
- {
- return one two;
- }
- }

Constructor

- The default constructor creates or instantiates a new object of the a class, that is, the default constructor transparently allocates memory for each object of a class. The access modifier for the default constructor is usually public. Additionally, the default constructor executes the statements enclosed by the curly braces.
- Another name for the default constructor is the *no-argument* constructor.
- At the time of calling constructor, memory for the object is allocated in the memory.
- You can have default (no-args) or argument constructor

Static

- There can be;
  - Static nested class
  - Static block
  - Static variable (class variable)
  - Static method (class method)

Static Variables

- In general, if a class contains a static variable,
  - all objects/instances of the class share that variable;
  - there is only one variable or storage location allocated to the whole class;
  - the variable belongs to the class and not to any particular object;
  - the variable exists regardless of whether or not any objects have been created; and
  - the variable may be accessed using either the class name or an object name, if an object has been created.

Static Methods

- A static method exists whether or not any object exists.
- A static method may be called whether or not any object of the class exists.
- A static method may be called whether or not an object of the class exists, but a static method cannot invoke an instance method except via an object.
- Also, instance variables are not accessible within static methods.

The Keyword “this”

- The reference “this” refers to the current instance of a class, the object currently being used.
- By using this , an object can refer to itself.
- Because this refers to an object, the keyword this cannot be used in a static method because static methods can execute even if no objects have been created; however, this can be used in any non- static method.
- Using the keyword this, one constructor can call another constructor. And not by using the class name. --> this();
- If one constructor calls another constructor, no other statements can precede that call.

Garbage Collection

- The Java Virtual Machine automatically reclaims all memory allocated to unreferenced objects for future use. In other words, if an object is no longer referenced and accessible, the memory allocated to that object is freed and made available for the creation of other objects. This clean-up process is called *garbage collection*.
- A *memory leak* occurs when an application maintains references to obsolete objects.
- Obsolete objects are those objects which are referenced in the program but are no longer used in the program. The garbage collector will not reclaim the memory for those objects.
- objectReference = null; to reclaim memory for referenced objects.

Garbage Collection

- The Java constant null can be assigned to a reference. A reference with value null refers to no object and holds no address; it is called a *void reference*.
- As the reference variables are assigned null value, now these objects are no longer reference and memory is automatically reclaimed from these unreferenced objects.
- Managing memory use is an important part of a programmer's job. The programmer must work in tandem with Java's automatic garbage collection to ensure that there are no memory leaks.