Classes & Objects

Classes & Objects

- What is Class ?
- What is Object ?
- A simple class
- Creating Objects
- Assigning Object Reference Variables (Emp e1 = e2)
- Methods (With paratmeters and return)
- Constructors
- finalize

Classes & Objects

- Parameterized Constructors
- “this” keyword
- Instance variable hiding / Variable Shadowing
- Garbage Collection
- finalize method
- method overloading
- Constructor overloading

Classes & Objects

- Using Objects as Parameters
- Argument passing (Call by value and call by reference)
- Returning objects
- Static member of class
- Introducing Access Control
- final keyword and its use

Classes & Objects

Object

- An entity which does exist, has state and behavior is known as an object e.g. chair, bike, marker, pen, table, car etc.
- If something does not really exist, then it is not an object e.g. our thoughts, imagination, plans, ideas etc.,
- According to System existence means contains memory. So a software object represent a memory.
- Word object and instance used interchangeably.

Classes & Objects

Class

- It is possible that some objects may have similar properties and actions. Such objects belong to same category called a ‘class’
- It is only a logical component and not the physical entity e.g. If we have class of “Expensive Cars” it could have objects like Mercedes, BMW, Toyota, etc.
- Properties of car are price, speed etc.
- Class defines a new data type. Class is a template for an object and an object is an instance of a class.

Classes & Objects

Class

- It is possible that some objects may have similar properties and actions. Such objects belong to same category called a ‘class’
- It is only a logical component and not the physical entity e.g. If we have class of “Expensive Cars” it could have objects like Mercedes, BMW, Toyota, etc.
- Properties of car are price, speed etc.

Classes & Objects

Class declaration syntax

<modifier> class <ClassName>{

//Class body with variables and methods

}

<modifier> => public, default (not a keyword), final, strictfp (For all classes)

<modifier> => public, default (not a keyword), private, protected, final, strictfp, static (For inner classes)

Classes & Objects

Class modifiers

- A class may be Access modifiers: public, protected, and private
- Modifier requiring override: abstract
- Modifier restricting to one instance: static
- Modifier prohibiting value modification: final
- Modifier forcing strict floating point behavior: strictfp
- declared with one or more modifiers which affect its runtime behavior.

Classes & Objects

Class Creation: Example.

```
class Cricketer
{
        //Data/Properties
        String name;
        String country;
        int totalMatches;
        // Methods/Behaviours/functions
        void getDetails();
        void displayDetails();
}
```

Classes & Objects

Constructor

- A constructor initializes an object immediately upon creation. Once defined, the constructor is automatically called immediately after the object is created before the new operator completes.
- The implicit return type of a class' constructor is the class type itself.
- It constructs the values i.e. provides data for the object that is why it is known as constructor.
- Name of the constructor must be same as that of a class name.
  Constructors must not have a return type. If we keep return type for the constructor, it will be treated as another method.
- If we don't write any constructor, compiler gives default constructor.
- Constructor can be declared private. In this case it can not be accessed from outside the class.

Classes & Objects

Constructor

- There can be multiple overloaded constructor inside a class.
- Only public, private, protected keyword are allowed before any constructor name. Applying any other keywords (like static, final etc.) will give compilation error.
  Ex. **static** Cricketer() { } // **Error**

Classes & Objects

‘this’ keyword

- Reference to the current object. That is, this is always a reference to an object on which method was invoked.
- ‘this’ can be used to call the overloaded constructor from the other constructors within the same class.
  But it should always be the first statement within the constructor.
  Ex.

```
Cricketer(int totRun){
        this();                 //Calling no argument constructor of Cricketer class.
        //Other code
```

Classes & Objects

- this ( ) can not be used within any other methods other than constructors

```
void updateTotalRuns(int totRuns)
{
        //this();       //Compilation Error,  this must be first statement in constructor
        totalRuns = totRuns;
}
```

Classes & Objects

Variable Shadowing

- When a local variable has the same name as an instance variable, the local variable hides the instance variable. We can use ‘this’ to resolve this name space collisions.

```
void updateTotalRuns(int totalRuns)
{
    //this();                  //Compilation Error,  this must be first statement in constructor
    this.totalRuns = totalRuns; // Resolving variable shadowing
}
```

Classes & Objects

Method Overloading

- Two or more methods within the same class that share the same name with different parameter list.
- The overloaded methods must differ in the type and/ or number of their parameters.
- Overloaded methods can have different return types.
- Constructors can also be overloaded.

Classes & Objects

Different Type of Variables used inside Class

- static variable
- Instance variable
- local variable

Classes & Objects

Instance variable

- Variables that are part of each object or we can say each instance of class contains its own copy of these variables.
- Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null.
- Values can be assigned **during the declaration** or within the **constructor**.

Local variable

- A variable which is declared inside the methods, constructors, or blocks is called local variable.

Classes & Objects

Static variable

- Class variables also known as static variables are declared with the static keyword in a class.
- There would only be one copy of each class , regardless of how many objects are created from it
- Default values are same as instance variables. Values can be assigned **during the declaration** or within the **constructor**. Additionally, values can be assigned in special **static initializer blocks**.
- Static variables can be accessed by calling the class name ClassName.VariableName.

Classes & Objects

Static block

- Static block is used for initializing the static variables.
- A static keyword is prefixed before the start of the block.
- This block gets executes only once when the class is loaded in the memory.
- A class can have multiple Static blocks, which will execute in the same sequence in which they have been written into the program.
- Inside static block all static variables can be accessed freely
- Instance variables can also be accessed but only through object reference after object creation.
- Syntax:

```
Class ABC{
        static
        {
                //static block
        }
}
```

Classes & Objects

Non-static block

- A non-static block executes when the object is created, before the constructor.
- Unlike static block, no keyword is prefixed before the start of the block.
- This block gets executed every time when any object of the class is created.
- A class can have multiple non-static blocks, which will execute in the same sequence in which they have been written into the program.
- Inside non-static block all static and non-static variables can be accessed freely
- Syntax:

```
Class ABC{
        {
            //non-static block
        }
}
```

Classes & Objects

Use of ‘static’ keyword within class

- static keyword can be applied on data member as well as member functions.
- When a class member is declared static, it can be accessed before any objects of its class are created, and without reference to any object using class-name.static-member

Static data member

- When objects of a class are declared/created, no copy of a static variable is made. Instead all instances of the class share the same static variable.

static member functions

Methods declared as static have several restrictions

- *They can only call other static methods*
- *They can only access static data*

Classes & Objects

Call by value & Call by reference

- Primitive type data are passed by value.
- Object are passed by using reference but that reference is passed by value.