



# OOP in JAVA



Classes ●  
Objects ●  
Methods ●  
Interfaces ●

By,

Hitha Paulson

Assistant Professor, Dept. of Computer Science

LF College, Guruvayoor



# OOP Concept

- Everything in OOP is considered as Classes and Object
- Prime importance is given to Data
- Identify data in the system
- Group data based on the common property
- Develop a template for the data group
- Class is a template for data
  - Class is a data type
  - Class does not contain any data
  - Class contain data variables and methods to operate data
- Object is an instance of Class
  - Objects are created from Class

# General form of CLASS

```
class classname
{
type instance-var1;
type instance-var1;
// .....
type instance-varN;
type methodname1(parameter-list)
{
    //body of method
}
type methodname2(parameter-list)
{
    //body of method
}
// .....
type methodnameN(parameter-list)
{
    //body of method
}
}
```

# About CLASS definition

- Declared by using **class** keyword
- Variables defined within class are called **instance variable**
- Code is written within **methods**
- Instance variables and methods together called **class members**
- No method definition can put outside class
- With each member, it is required to use access-specifier
- The default access specifier is FRIENDLY

# Sample Class

```
class rectangle
{
int length, breadth;

void getData(int x, int y)
{
    length = x;
    breadth = y;
}
int rectArea()
{
    int area = length * breadth;
}
}
```



# Creating Objects

- Objects are created in two steps
  - Declare a class type variable
  - Allocate memory and bind with class variable
  - Memory is allocated dynamically by using **new** operator
- Eg: `rectangle r1; //declaring class variable`  
`r1 = new rectangle(); //allocating memory and binding`  
`rectangle r2= new rectangle();`
- `rectangle()` : call to constructor
- Class members are accessed by using dot (.) operator
- Eg: `r1.getData(10,20); r1.length=45;`

# Constructors

- Member functions used to initialize class data members
- Constructors has the same name of Class
- A class can have more than one constructor
- Matching constructor function will invoke each time when an instance is created
- Constructors will not have any return value, even void
- Implicit return type of constructor is **class type** itself

# Parameter Constructors

- Constructors can take parameters
- Parameters are passed at the time of object creation
- Once parameterized constructors are there, all object creation is required to use proper number of parameters
- Default constructor is required to use, if you want to initialize object without parameter



# this Keyword

- It is keyword that can use inside a class method
- Used to refer to the object that invoked the method

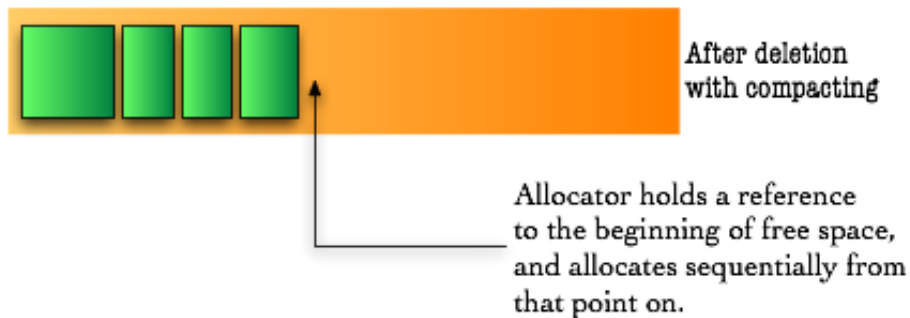
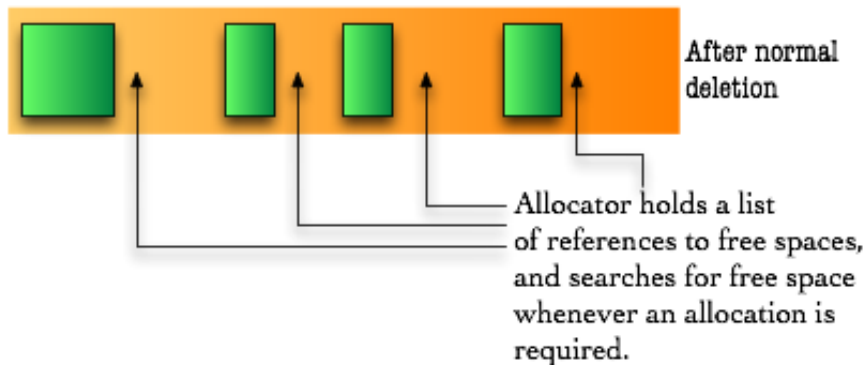
## Hiding instance variable




- When we use formal variables or local variables having the same name of INSTANCE variables, then instance variables will be hidden
- this keyword can use to override instance variable hiding
- this.INSTANCE variable-name refer to INSTANCE variable

# Garbage Collection

- Garbage collection is the process of recollecting the memory that is no longer used
- This process is automatically and dynamically done by JAVA
- When no reference to an object exist, object is assumed to be no longer needed, and the memory occupied by the object will automatically reallocated
- helps to ensure program integrity
- Disadvantage: Takes lot of processing to find out unreferenced objects

# Garbage Collection Scenario



-  A live object
-  Marked for deletion
-  Memory space

# finalize() method

- It is an alternate to the destructor function
- Used to deallocate non-Java resources held by a Java program
- Non-Java resources Eg: File handler, Window character font
- This method will be called automatically just before the garbage collection
- It is declared as protected to prevent access to finalize() by code defined outside the class
- finalize() will not be invoked when an object goes out of scope

# Method Overloading

- A class can have more than one method having the same name, they are Overloaded methods
- Overloaded methods are different in terms of number of OR type of parameters
- Method overloading is one of the ways that Java implements Polymorphism
- Difference in return type is not sufficient for method overloading
- Constructors can also overload

# Objects as Parameters

- Methods can take Objects as its parameter
- Constructors can also take Object as argument
- Object type argument are used to make available the data in one object into another object
- Objects can return from method

## Parameter passing

- Call-by-Value :- Copies the value of actual argument into formal argument. Changes in formal argument will not reflect in actual argument
- Call-by-Reference :- Reference to the parameter is passed as argument. Changes in formal argument in reflect in actual argument
- Object type parameters are passed as Call-by-reference

# Recursion

- A method that calls itself is said to be **recursion**
- Local variables and formal arguments of recursive function will create every time when function call is made
- When a function call returns, corresponding local variables and parameters will destroy
- One of the possible exception that can raise is **STACK OVERFLOW**
- It is mandatory to use some condition to terminate recursive function call



# STATIC members

- Members of class declared by using the keyword **static** are known as static members
- Data members and methods can be static
- Static Data member
  - It is global variable to the class
  - All objects of class share same Static member
  - No copy of static variable is created for each object
  - Static member are accessed with class name but not with objects of class
  - It is created before any object of that class is created



# STATIC members contd..

- **Static Method**

- They can access only static data members
- They can call only other static methods
- Static methods are accessed with class name (classname.staticmethod)
- Eg: main() method:- This function will invoke without creating object of class
- They cannot refer to "this" or "super" because static methods are not object specific

# FINAL variables

- FINAL variables are constants in JAVA
- FINAL variables are declared by using **final** keyword
- All final variables are required to initialize at the time of declaration
- Eg: `final int count = 100;`
- Value of final variable cannot be changed during execution



# Nested and Inner Classes

- A class defined inside another class is known as **Nested** class
- If class B is defined inside class A, then A is outer class and B is Nested class
- Nested class is available within the scope of its enclosing class
- Nested class can access the members including private members of the class in which it is enclosed
- Enclosing class cannot directly access the members of nested class
- Nested class can define within the scope of a block bracket in another class



# Nested and Inner Classes contd..

- Nested class can be STATIC or NON-STATIC
  - Nested class can be either static or non-static
  - Non static Nested class is called **Inner Class**
  - Inner class can access variables and methods of outer class without using objects
  - Instance of inner class can create only within the scope of outer class
  - Eg: `outerclass.innerclass`
  - Anonymous inner classes are inner classes that don't have name

# Inheritance

- Inheritance is the process of deriving the properties of one class into another class
- A class that is derived from another class is called a subclass (also a derived class, extended class, or child class).
- The class from which the subclass is derived is called a superclass (also a base class or a parent class).
- Inheritance helps to add more properties to existing class and hence reuse existing classes
- When inheritance is performed, all the inheritable members of super class will be available in derived class
- There are different forms of Inheritance. Eg: Single, Hierarchical, Multilevel

# Inheritance in Java

- Inheritance is done by using **extends** keyword
- Java does not support Multiple inheritance(a class **extended** from more than one classes)
- Private members of a class cannot inherit
- By default all the non-private members including constructors will be available in derived class
- General form

```
class subclass-name extends superclass-name
{
//body of class
}
```

# Objects of Superclass and Subclass

- By using the objects of Subclass we can access all members of Superclass and Subclass
- By using the objects of Superclass we can access only the members of Superclass
- We can assign an object of subclass into object of superclass. Here also object of superclass can access members of Superclass only.

# super Keyword

- This keyword is used to refer the immediate superclass from a subclass
- super has two general forms
  - Used to call superclass constructor
  - Used to access a member of the superclass that has been hidden by a member of a subclass
- Cannot use it within static method



# Form 1

- Subclass can call a constructor method defined by its superclass
- General condition
  - It should be the first statement in the subclass constructor
  - If superclass contains constructor then subclass should contain a constructor
- Usage: `super(parameter-list);`
- Subclass constructor should contain enough arguments to call superclass constructor

## Form 2

- Used to access a member of the superclass that has been hidden by a member of a subclass
  - General form **super**.*member*; where member is a data member in superclass
  - **super** can use to refer only the immediate class member
  - **super** can also used to refer methods of superclass

# Constructor in Multilevel Hierarchy

- Constructors in a multilevel hierarchy is invoked in the order of their definition.
- Constructor of superclass will invoke before subclass constructor
- Once a hierarchy is defined with top class contains constructor, then all the classes in this hierarchy must contain constructors

# Method Overriding

- The process of defining a method in a subclass having the same name and type signature of a method in superclass
- When an overridden method is called from a subclass, it will always refer to the subclass version of the method
- In order to call superclass version of overridden method, **super** keyword can precede with method name
- Method overriding occurs only when the name and type signature of the two methods are identical, otherwise they are overloaded methods

# Dynamic Method Dispatch

- It is the process of implementing run-time polymorphism
- Method overriding helps to implement run-time polymorphism
- Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time
- When an overridden method is called through a superclass reference, Java determines which version of that method to execute based upon the type of object being referenced at the time of call occurs
- The type of object being referenced determines which version of overridden method will be executed

# Abstract Class

- A class declared by using **abstract** keyword is called Abstract class
- An abstract class contains Abstract Method and this class cannot instantiate using **new** operator
- An abstract method declared in a base class which never contains its definition
- Abstract methods must override in the derived class. Either redefine as abstract or else give its definition
- **abstract** keyword is used to declare abstract methods
- Abstract class cannot contain abstract constructor and it cannot be abstract static method
- Abstract class can use to declare Object References



# final Class and final Method

- A class declared by using **final** keyword is called final class
- A final class cannot be used to inheritance and all of its methods will be final methods
- A final class cannot be both **abstract** and **final**
- A method declared by using **final** keyword is called final method
- A final method cannot be override by subsequent classes
- Final methods ensure early binding