Saraswati Vandana

या कुन्देन्दु तुषार हार धवला
या शुभ्र वस्त्रान्विता ।
या वीणा वर दंड मंडितकरा
या श्वेत पद्मासना ॥

या ब्रह्मा अच्युत शंकर प्रभृतिभिः
देवै सदा पूजिता ।
सा मां पातु सरस्वती भगवती
निःश्येश जाड्यापह ॥

Faculty of Engineering & Technology
Sankalchand Patel College of Engineering, Visnagar

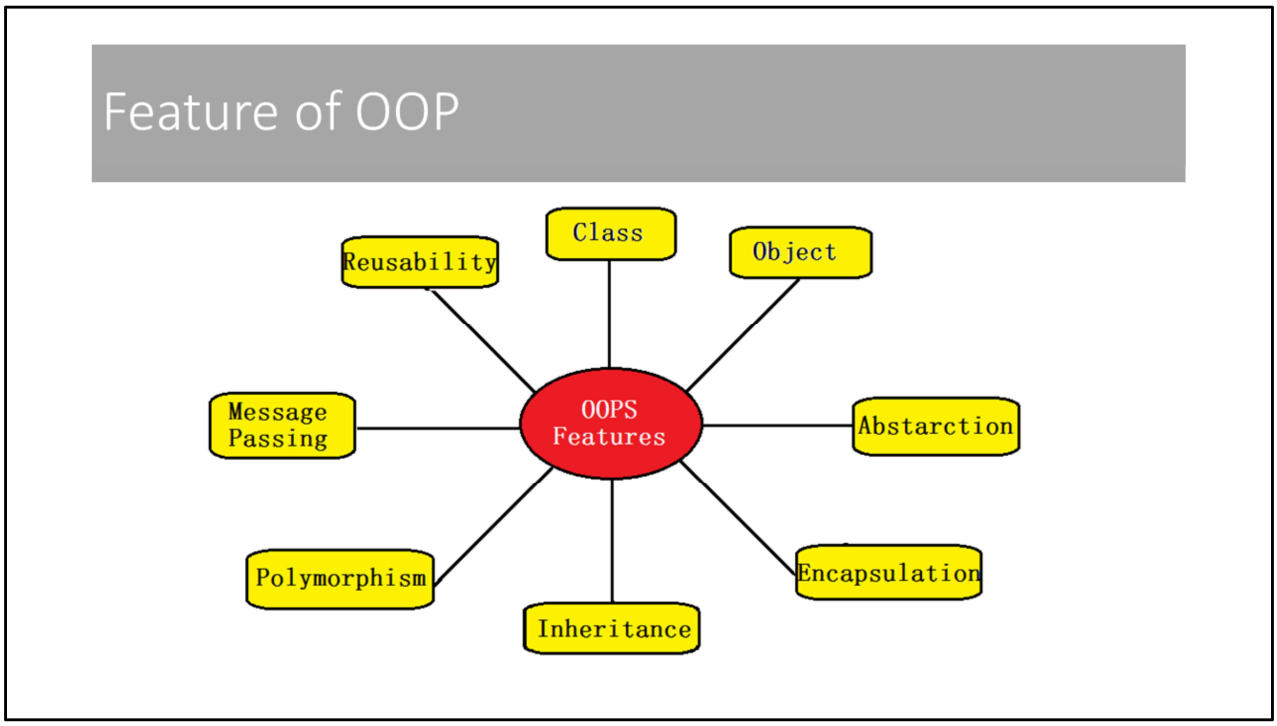
Java Programming
(1ET1030406)

Unit-3 : Classes, Objects and Methods

Prepared By
Mr. Mehul S. Patel
Department of Computer Engineering & Information Technology

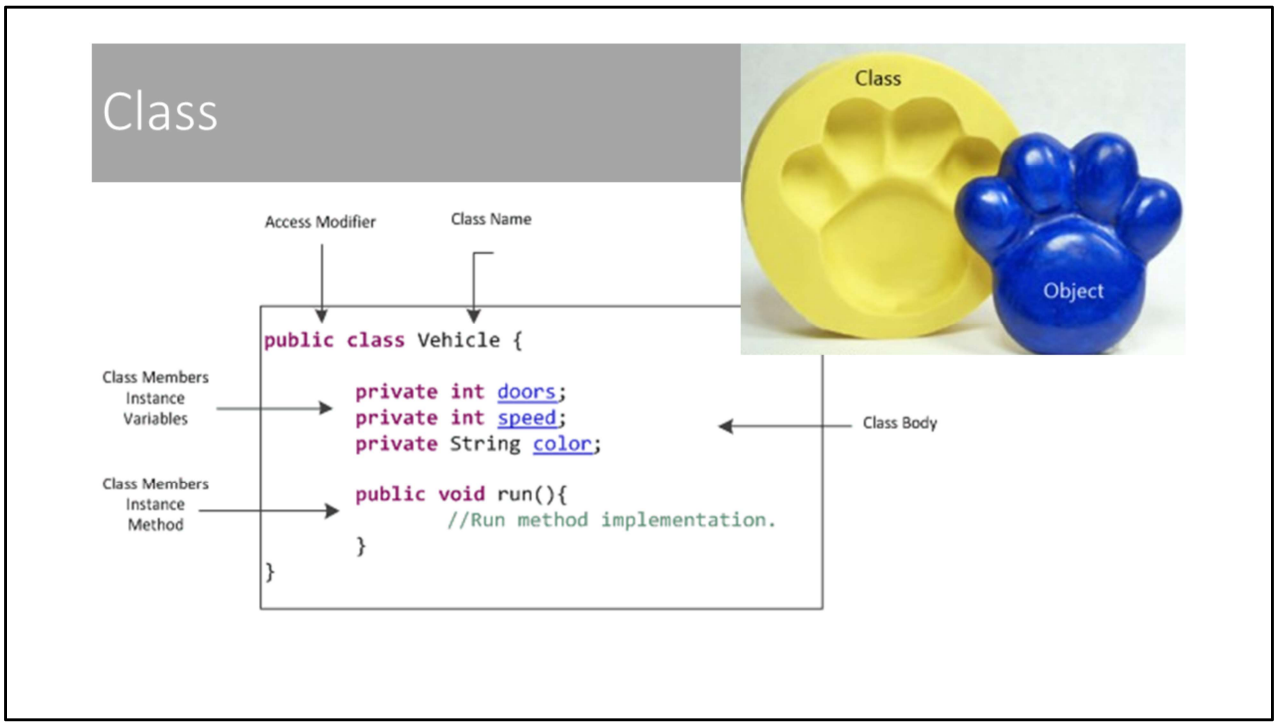
Content

- Feature of OOP
- Object reference
- Constructor
- Constructor Overloading
- Method Overloading
- Recursion
- Passing and Returning object form Method
- new operator
- this and static keyword
- finalize() method
- Access control modifiers
- Inner class (Nested class)
- Anonymous inner class



Features of OOP

Object-oriented programming System(OOPs) is a programming paradigm based on the concept of “objects” that contain data and methods. The primary purpose of object-oriented programming is to increase the flexibility and maintainability of programs. Object oriented programming brings together data and its behaviour(methods) in a single location(object) makes it easier to understand how a program works.

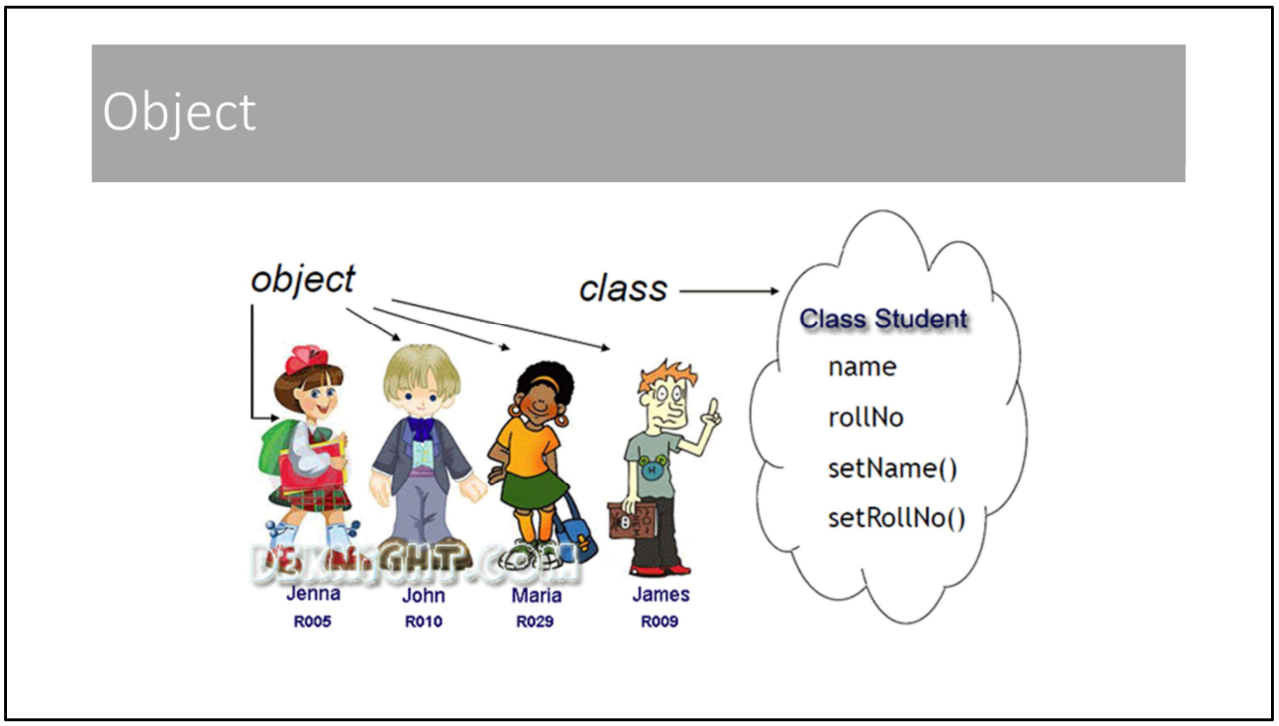


Class

- A class can be defined as a set of properties or methods that are common to all objects of one type.
- A class can be considered as a blueprint using which you can create as many objects as you like.
- Class is a logical entity to represents a real world entity
- a class can be defined as *a mold or template for creating objects.*
- In the above example, the class is the mold (in yellow color) and the object is the paw (blue color). All the objects (blue color) will have the same size, shape etc.

In general, class declarations can include these components, in order:

- **Modifiers**: A class can be public or has default access.
- **class keyword:** class keyword is used to create a class.
- **Class name:** The name should begin with an initial letter (capitalized by convention).
- **Superclass(if any):** The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
- **Interfaces(if any):** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
- **Body:** The class body surrounded by braces, { }.



Object

Object is an instance of a class. An object in OOPS is nothing but a self-contained component which consists of methods and properties to make a particular type of data useful. For example color name, table, bag, barking. When you send a message to an object, you are asking the object to invoke or execute one of its methods as defined in the class.

From a programming point of view, an object in OOPS can include a data structure, a variable, or a function. It has a memory location allocated. Java Objects are designed as class hierarchies.

It is a basic unit of Object-Oriented Programming and represents the real life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of :

State: It is represented by attributes of an object. It also reflects the properties of an object.

Behavior: It is represented by methods of an object. It also reflects the response of an object with other objects.

Identity: It gives a unique name to an object and enables one object to interact with other objects.

Declaring Objects

As we declare variables like (type name;). This notifies the compiler that we will use name to refer to data whose type is type. With a primitive variable, this declaration also reserves the proper amount of memory for the variable. So for reference variable, type must be strictly a concrete class name. In general, we **can't** create objects of an abstract class or an interface.

Like Student st;

If we declare reference variable(st) like this, its value will be undetermined(null) until an object is actually created and assigned to it. Simply declaring a reference variable does not create an object.

Initializing an object

The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the class constructor.

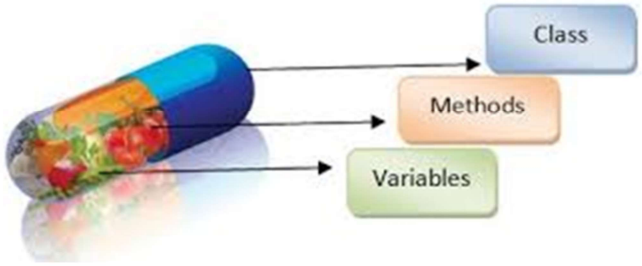
Like st = new Student( );

Abstraction

Abstraction

- *Hiding internal details and showing functionality* is known as abstraction. For example phone call, we don't know the internal processing.
- In Java, we use abstract class and interface to achieve abstraction.
- Abstraction is a process where you show only “relevant” data and “hide” unnecessary details of an object from the user.
- In OOP, abstraction means hiding the complex implementation details of a program, exposing only the API required to use the implementation.

Encapsulation



Encapsulation

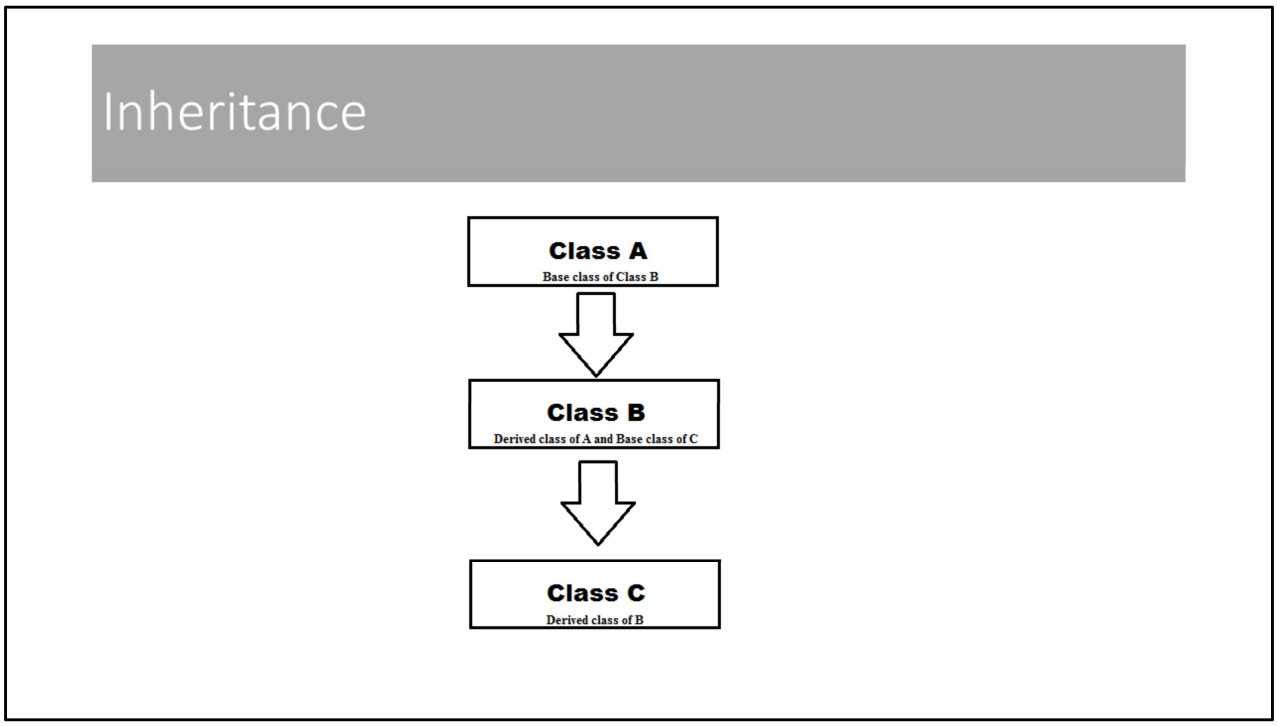
Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit, for example, a capsule which is mixed of several medicines.

We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

The **Java Bean** class is the example of a fully encapsulated class.

Advantage of Encapsulation in Java

- By providing only a setter or getter method, you can make the class **read-only or write-only**. In other words, you can skip the getter or setter methods.
- It provides you the **control over the data**. Suppose you want to set the value of id which should be greater than 100 only, you can write the logic inside the setter method. You can write the logic not to store the negative numbers in the setter methods.
- It is a way to achieve **data hiding** in Java because other class will not be able to access the data through the private data members.
- The encapsulate class is **easy to test**. So, it is better for unit testing.
- The standard IDE's are providing the facility to generate the getters and setters. So, it is **easy and fast to create an encapsulated class** in Java.



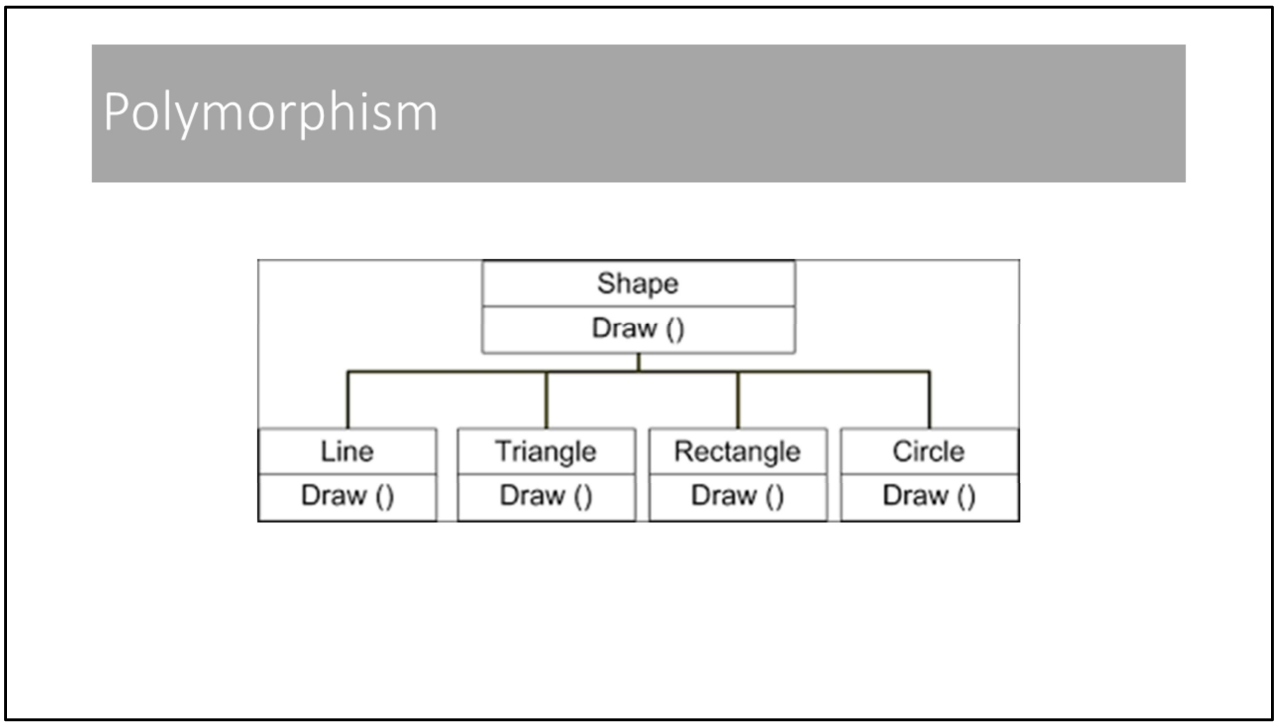
Inheritance

It's nice to have a bunch of objects and their collective methods and attributes, but what would be nicer would be for us to share these methods and attributes across other related objects, and why would we want to do this? Well, mostly for **re-usability** and maintaining the DRY (Do not Repeat Yourself) principle. This is where Inheritance comes in handy, inheritance allows one class (**subclass/child**) "inherit" the attributes and methods from another class (**super-class/parent**).

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object.

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.



Polymorphism in Java

Polymorphism in Java is a concept by which we can perform a *single action in different ways*. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

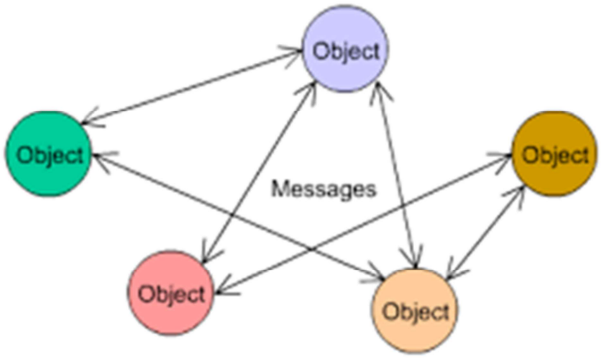
There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

If you overload a static method in Java, it is the example of compile time polymorphism. Here, we will focus on runtime polymorphism in java.

Runtime polymorphism or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

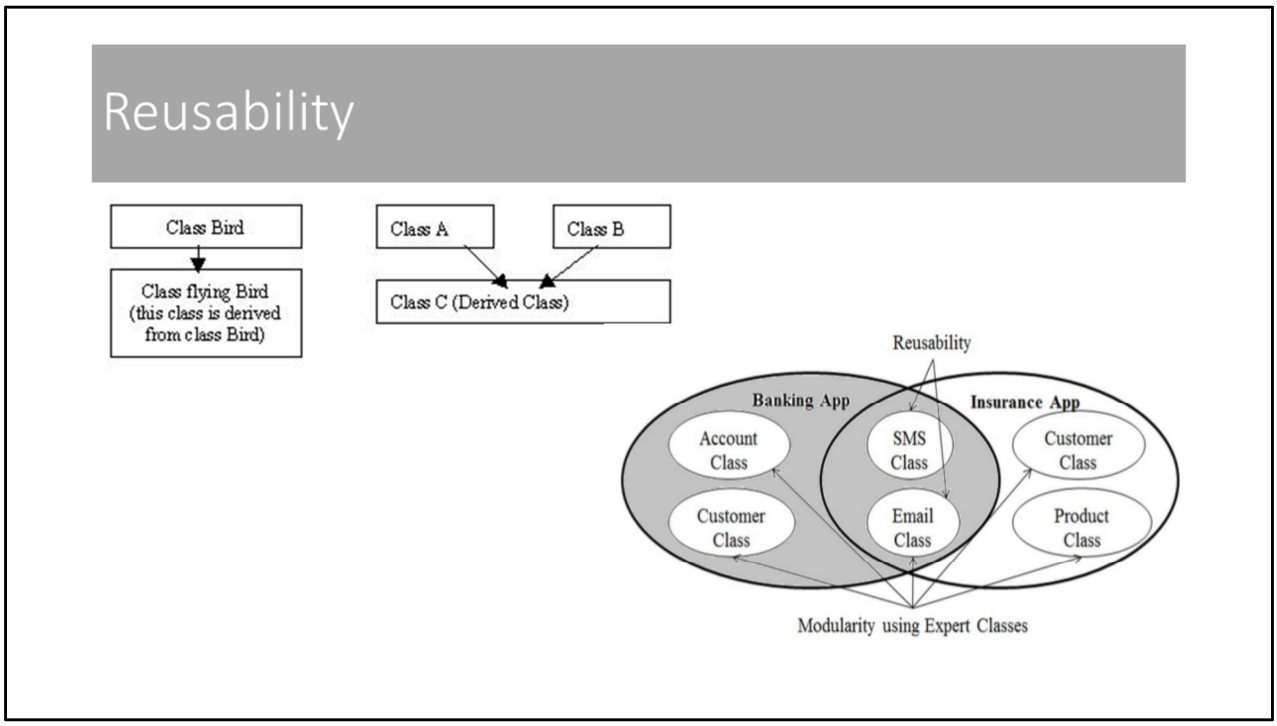
Message Passing



Interaction of objects via message passing

Message Passing

- Object-oriented programming as a programming paradigm is based on objects. Objects are a representation of real-world and objects communicate with each other via messages.
- When two or more objects communicate with each other that means that those objects are sending and receiving messages. This is often called method calling.
- Sending object (Object A) knows which method of receiving object (Object B) is being called, so it knows more details than needed. With this, we create code which is not loosely coupled. Changes in one object will lead to changes in others too.



Reusability

Well-designed classes are software components that can be reused without editing. A well-designed class is not carefully crafted to do a particular job in a particular program. Instead, it is crafted to model some particular type of object or a single coherent concept. Since objects and concepts can recur in many problems, a well-designed class is likely to be reusable without modification in a variety of projects.

Furthermore, in an object-oriented programming language, it is possible to make subclasses of an existing class. This makes classes even more reusable. If a class needs to be customized, a subclass can be created, and additions or modifications can be made in the subclass without making any changes to the original class. This can be done even if the programmer doesn't have access to the source code of the class and doesn't know any details of its internal, hidden implementation.

Way to reuse

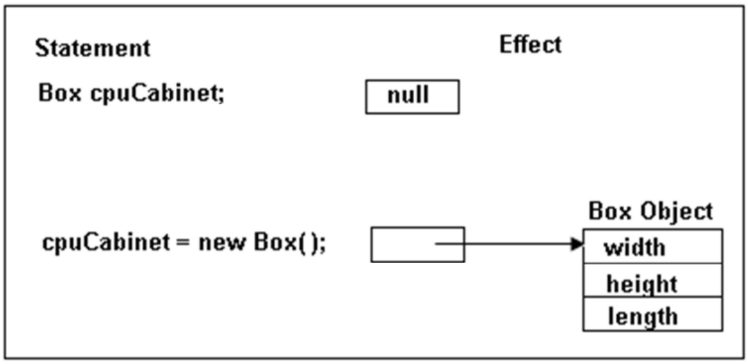
1) Association (has-A)

i)composition(Strong)

ii)Aggregation(Weak)

2) Inheritance (is-A)

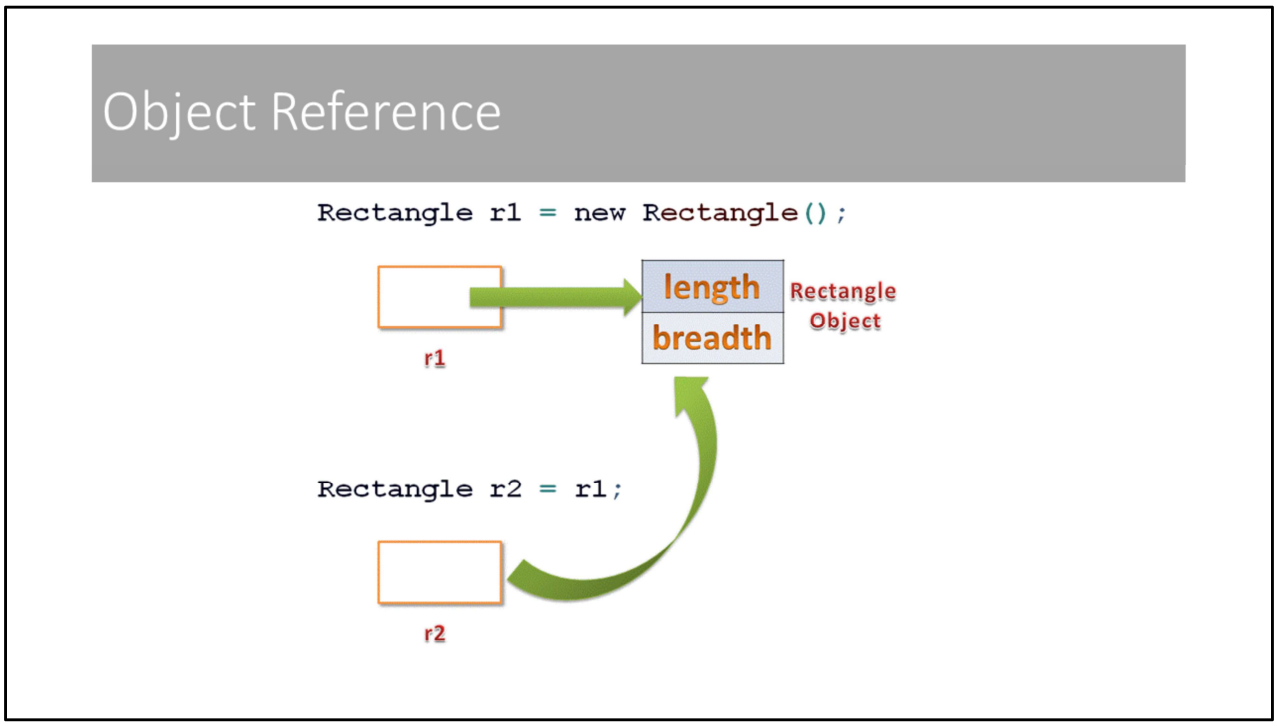
new Operator



new Operator

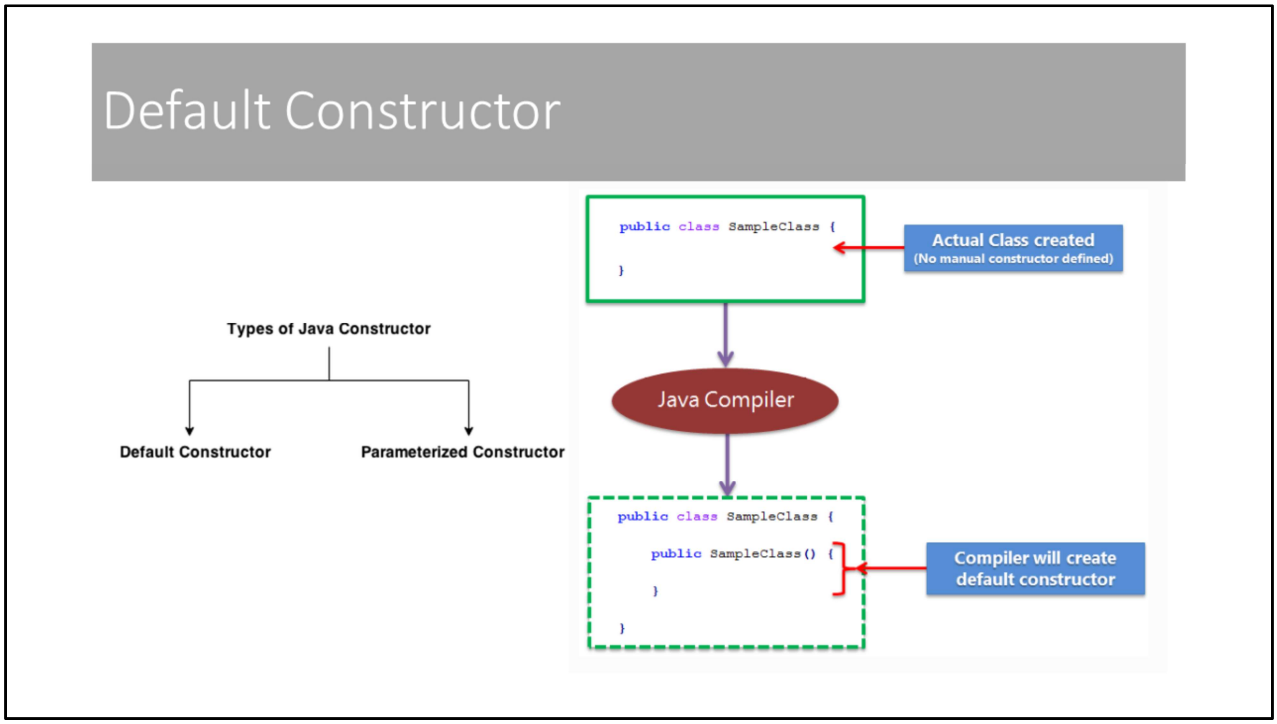
The ***new*** operator instantiates a class by dynamically allocating(i.e, allocation at run time) memory for a new object and returning a reference to that memory. This reference is then stored in the variable. Thus, in Java, **all class objects must be dynamically allocated.** The ***new*** operator is also followed by a call to a class constructor, which initializes the new object.

- The new operator is used in Java to create new objects.
- It can also be used to create an array object.



Object Reference

- A *reference* is an address that indicates where an object's variables and methods are stored.
- You aren't actually using objects when you assign an object to a variable or pass an object to a method as an argument. You aren't even using copies of the objects. Instead, you're using references to those objects.



Default Constrictor

If we do not create any constructor, the Java compiler automatically create a no-arg constructor during the execution of the program. This constructor is called default constructor.

Need of Constructor

Think of a Box. If we talk about a box class then it will have some class variables (say length, breadth, and height). But when it comes to creating its object(i.e Box will now exist in computer's memory), then can a box be there with no value defined for its dimensions. The answer is no. So constructors are used to assign values to the class variables at the time of object creation, either explicitly done by the programmer or by Java itself (default constructor).

When is a Constructor called ?

Each time an object is created using **new()** keyword at least one constructor (it could be default constructor) is invoked to assign initial values to the **data members** of the same class.

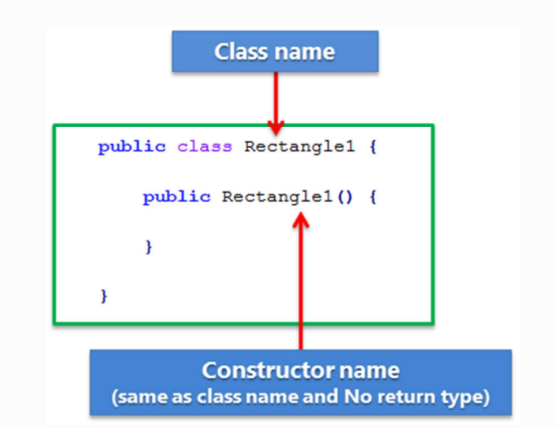
Rules for writing Constructor:

- Constructor(s) of a class must have same name as the class name in which it resides.
- A constructor in Java can not be abstract, final, static and Synchronized.
- Access modifiers can be used in constructor declaration to control its access i.e which other class can call the constructor.

Types of constructor:

1. No-argument constructor
2. Parameterized Constructor

Without Parameter Constructor

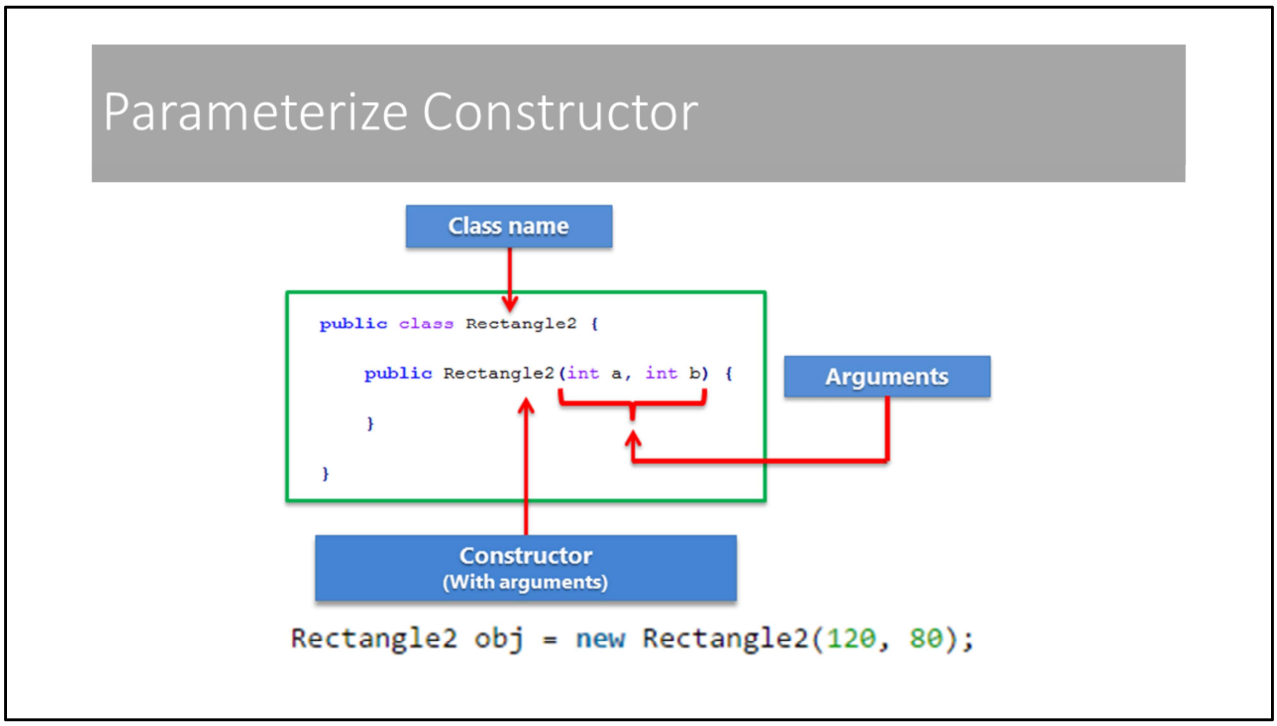



```
Rectangle1 obj=new Rectangle1();
```

No-argument constructor:

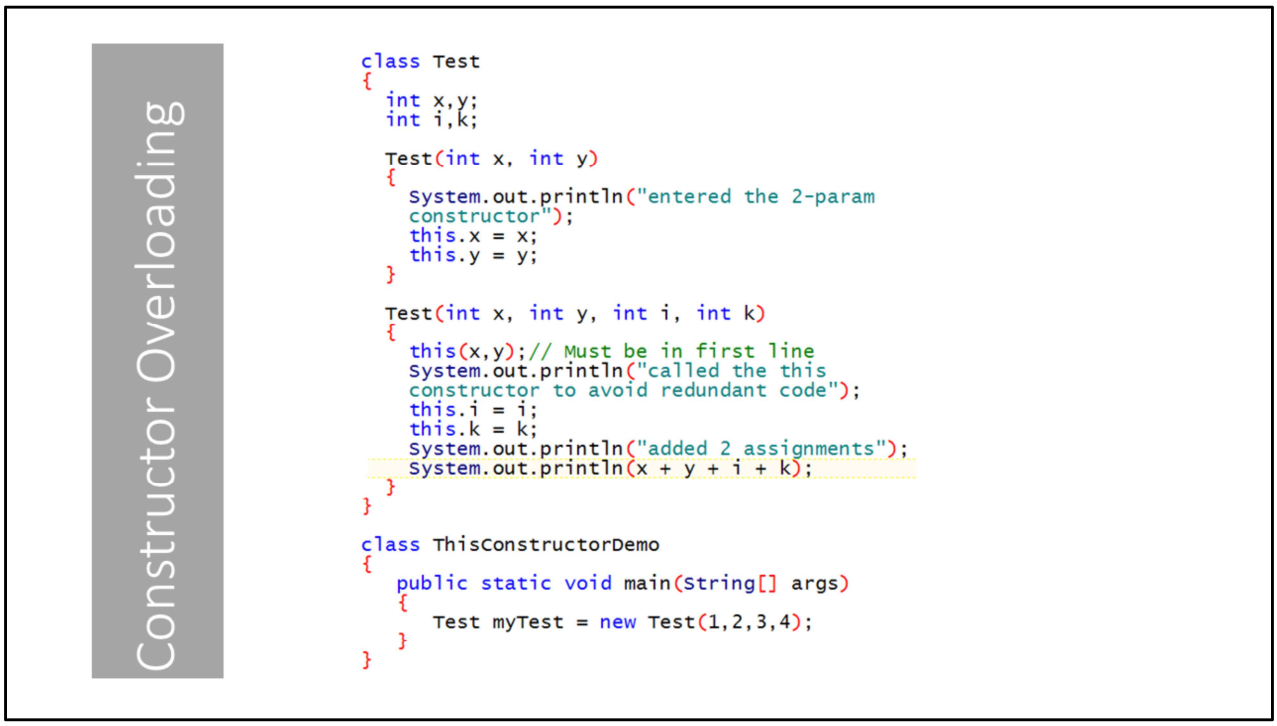
A constructor that has no parameter is known as default constructor. If we don't define a constructor in a class, then compiler creates **default constructor(with no arguments)** for the class. And if we write a constructor with arguments or no-arguments then the compiler does not create a default constructor.

Default constructor provides the default values to the object like 0, null, etc. depending on the type.



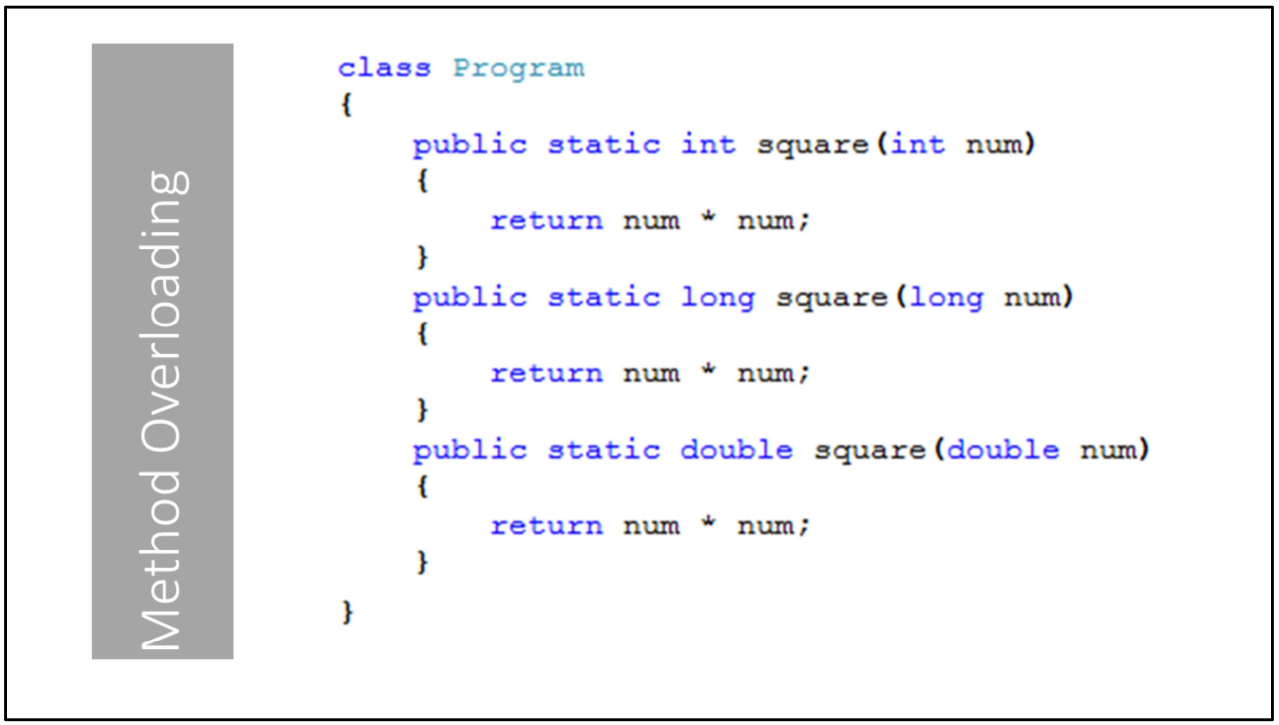
Parameterized Constructor:

A constructor that has parameters is known as parameterized constructor. If we want to initialize fields of the class with your own values, then use a parameterized constructor.



Constructor Overloading - Multiple Constructors for a Java Class

A class can have multiple constructors, as long as their signature (the parameters they take) are not the same. You can define as many constructors as you need. When a Java class contains multiple constructors, we say that the constructor is overloaded (comes in multiple versions). This is what *constructor overloading* means, that a Java class contains multiple constructors.



Method Overloading in Java with examples

Method Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different. It is similar to constructor overloading in Java, that allows a class to have more than one constructor having different argument lists.

when we say argument list it means the parameters that a method has: For example the argument list of a method add(int a, int b) having two parameters is different from the argument list of the method add(int a, int b, int c) having three parameters.

Three ways to overload a method

In order to overload a method, the argument lists of the methods must differ in either of these:

1. Number of parameters.

   For example: This is a valid case of overloading

   add(int, int) , add(int, int, int)

2. Data type of parameters.

   For example:

   add(int, int) , add(int, float)

3. Sequence of Data type of parameters.

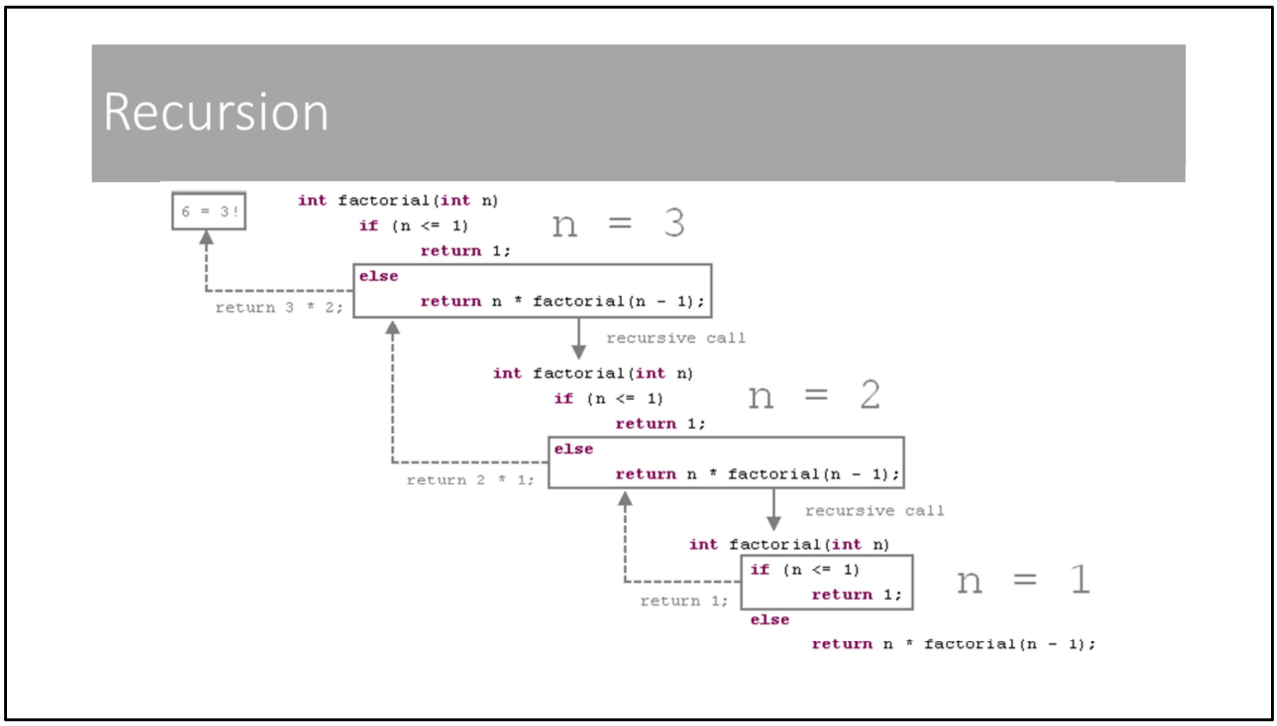
   For example:

   add(int, float), add(float, int)

Invalid case of method overloading:

When we say argument list, its 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)



Recursion

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function. Using recursive algorithm, certain problems can be solved quite easily. Examples of such problems are Towers of Hanoi (TOH), Inorder/Preorder/Postorder Tree Traversals, DFS of Graph, etc.

base condition in recursion

In the recursive program, the solution to the base case is provided and the solution of the bigger problem is expressed in terms of smaller problems. In the above example, base case for n < = 1 is defined and larger value of number can be solved by converting to smaller one till base case is reached.

How a particular problem is solved using recursion?

The idea is to represent a problem in terms of one or more smaller problems, and add one or more base conditions that stop the recursion. For example, we compute factorial n if we know factorial of (n-1). The base case for factorial would be n = 0. We return 1 when n = 0.

Why Stack Overflow error occurs in recursion?

If the base case is not reached or not defined, then the stack overflow problem may arise.

Passing and Returning object from Method

```
class Student {
String name;
float spi;
Student(String name,float spi)
{
    this.name=name;
    this.spi=spi;
}
Student higher(Student s)
    {
    if(this.spi>s.spi)
        return this;
    else
        return s;
    }
void print()
{
    System.out.println("Name: " + name);
    System.out.println("SPI: " + spi);
}
}
```

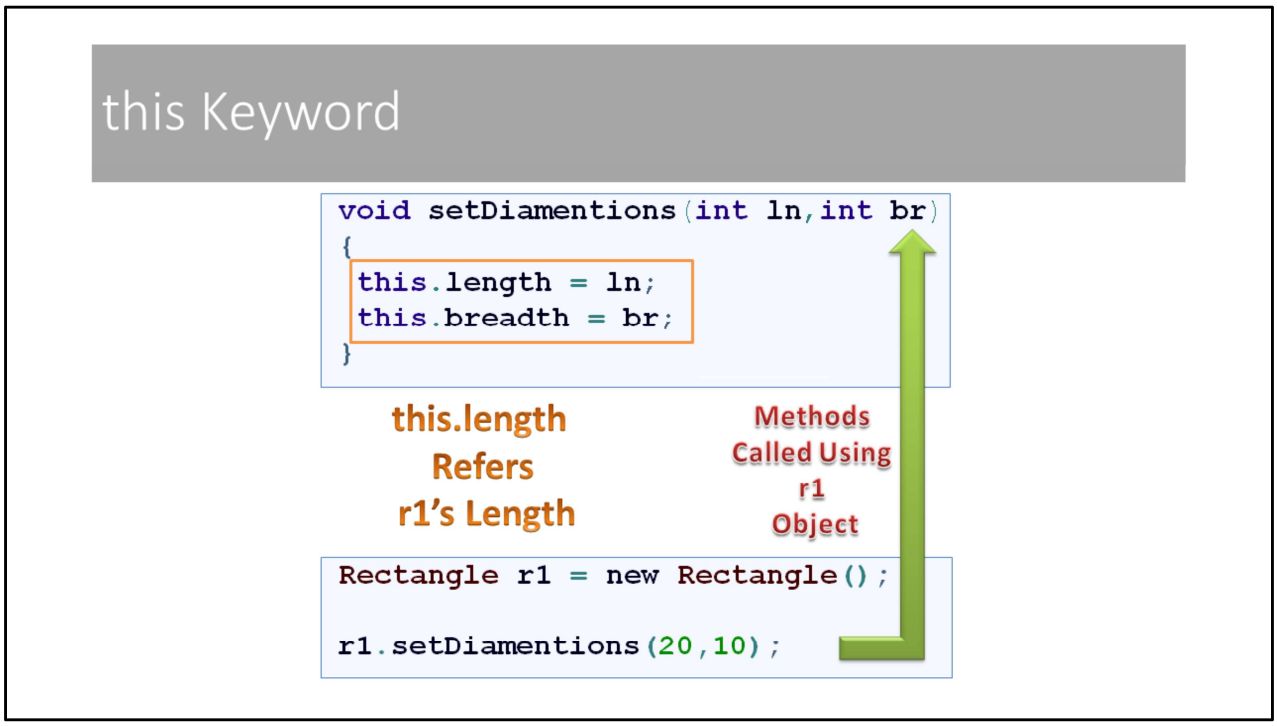
```
class Demo
{
    public static void main(String args[])
    {
        Student s1=new Student("abc",6.7f);
        Student s2=new Student("pqr",8.5f);
        Student temp= s1.higher(s2);
        temp.print();
    }
}
```

Passing and Returning Objects

Although Java is strictly pass by value, the precise effect differs between whether a primitive type or a reference type is passed.

When we pass a primitive type to a method, it is passed by value. But when we pass an object to a method, the situation changes dramatically, because objects are passed by what is effectively call-by-reference. Java does this interesting thing that's sort of a hybrid between pass-by-value and pass-by-reference. Basically, a parameter cannot be changed by the function, but the function can ask the parameter to change itself via calling some method within it.

- While creating a variable of a class type, we only create a reference to an object. Thus, when we pass this reference to a method, the parameter that receives it will refer to the same object as that referred to by the argument.
- This effectively means that objects act as if they are passed to methods by use of call-by-reference.
- Changes to the object inside the method do reflect in the object used as an argument.
- a method can return any type of data, including objects.



this Keyword

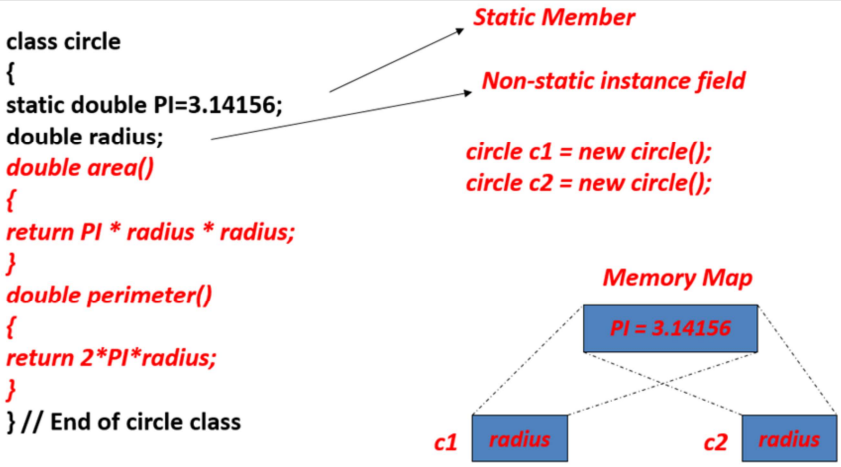
The this keyword refers to the current object in a method or constructor.

The most common use of the this keyword is to eliminate the confusion between class attributes and parameters with the same name (because a class attribute is shadowed by a method or constructor parameter). If you omit the keyword in the example above, the output would be "0" instead of "5".

this can also be used to:

- Invoke current class constructor
- Invoke current class method
- Return the current class object
- Pass an argument in the method call
- Pass an argument in the constructor call

static data member



The **static keyword** in Java is used for memory management mainly. We can apply static keyword with variables, methods, blocks and nested classes. The static keyword belongs to the class than an instance of the class.
The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)
3. Block
4. Nested class

static variable
If you declare any variable inside class as static, it is known as a static variable.

- The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
- The static variable gets memory only once in the class area at the time of class loading.
- It can be accessible by object or class name.

Advantages of static variable - It makes your program **memory efficient** (i.e., it saves memory).

static method

```
class BOX
{
private double l,b,h;   // Instance Fields
BOX(double a,double b,double c)
{
    l=a;this.b=b;h=c;
}       // Constructor
boolean isEqual(BOX other)
{
if (this.l == other.l &&
        this.b == other.b && this.h == other.h)
return true;
else
return false;
}
static boolean isEqual(BOX b1, BOX b2)
{
if (b1.l == b2.l && b1.b == b2.b && b1.h == b2.h)
return true;
else
return false;
}
} // End of BOX class
```

```
class statictest
{
public static void main(String args[])
{
BOX b1 = new BOX(10,6,8);
BOX b2 = new BOX(10,6,8);
BOX b3 = new BOX(1,16,18);
BOX b4 = new BOX(2,6,8);

System.out.println(b1.isEqual(b2));
System.out.println(BOX.isEqual(b1,b2));
System.out.println(b3.isEqual(b1,b2));
System.out.println(b4.isEqual(b2));
System.out.println(b4.isEqual(b4,b2));
}
}
```

static method

If you apply static keyword with any method, it is known as static method.

- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.

Restrictions for the static method

- There are two main restrictions for the static method. They are:
- The static method can not use non static data member or call non-static method directly.

this and super cannot be used in static context.

static Block

```
class StaticBlockDemo
{
    static
    {
        System.out.println("I am in static Block");
    }
    public static void main(String args[])
    {
        StaticBlockDemo obj= new StaticBlockDemo();
        StaticBlockDemo obj1= new StaticBlockDemo();
    }
}
```

static block

- It is used to initialize the static data member.
- It is executed before the time of class loading.

finalize() method

```
public class MyClass1 {
   @Override
   protected void finalize()
     {
          //....
     }

}
```

```
MyClass1 o1=new MyClass1();
o1.finalize();
```

finalize() method

The **finalize() method** of Object class is a method that the Garbage Collector always calls just before the deletion/destroying the object which is eligible for Garbage Collection, so as to perform clean-up activity. Clean-up activity means closing the resources associated with that object like Database Connection, Network Connection or we can say resource de-allocation. Remember it is not a reserved keyword. Once the finalize method completes immediately Garbage Collector destroy that object.

Syntax:

protected void finalize throws Throwable{}

Since Object class contains the finalize method hence finalize method is available for every java class since Object is the superclass of all java classes. Since it is available for every java class hence Garbage Collector can call the finalize method on any java object.

Why finalize method is used()?

finalize() method releases system resources before the garbage collector runs for a specific object. JVM allows finalize() to be invoked only once per object.

How to override finalize() method?

the finalize method which is present in the Object class, has an **empty implementation**, in our class clean-up activities are there, then we have to **override this method** to define our own clean-up activities.

Access Control Modifier

Visibility	Public	Protected	*Default*	Private
From the same class	Yes	Yes	Yes	Yes
From any class in the same package	Yes	Yes	Yes	No
From a subclass in the same package	Yes	Yes	Yes	No
From a subclass outside the same package	Yes	Yes, *through inheritance*	No	No
From any non-subclass class outside the package	Yes	No	No	No

Access Control Modifier

Access level modifiers determine whether other classes can use a particular field or invoke a particular method. There are two levels of access control:

- At the top level—public, or *package-private* (no explicit modifier).
- At the member level—public, private, protected, or *package-private* (no explicit modifier).

A class may be declared with the modifier public, in which case that class is visible to all classes everywhere. If a class has no modifier (the default, also known as *package-private*), it is visible only within its own package (packages are named groups of related classes — you will learn about them in a later lesson.)

At the member level, you can also use the public modifier or no modifier (*package-private*) just as with top-level classes, and with the same meaning. For members, there are two additional access modifiers: private and protected. The private modifier specifies that the member can only be accessed in its own class. The protected modifier specifies that the member can only be accessed within its own package (as with *package-private*) and, in addition, by a subclass of its class in another package.

Inner class (Nested Class)

```
public class Outer {
int a=10;
void print()
{
    Inner i=new Inner();
    i.print();
}
 class Inner
{int b=20;
   void print()
   {
       System.out.println(a+b);
   }
}
```

```
class Demo2
{
  public static void main(String a[])
{
    Outer o1=new Outer();
    Outer.Inner o2= o1.new Inner();

    o2.print();
}
}
```

Inner Classes

- **Java inner class** or nested class is a class which is declared inside the class or interface.
- We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.
- Additionally, it can access all the members of outer class including private data members and methods.

Advantage of java inner classes

There are basically three advantages of inner classes in java. They are as follows:

1) Nested classes represent a special type of relationship that is **it can access all the members (data members and methods) of outer class** including private.

2) Nested classes are used **to develop more readable and maintainable code** because it logically group classes and interfaces in one place only.

3) **Code Optimization**: It requires less code to write.

Anonymous inner class

```
public class Demostration {

    public static void main(String a[])
    {
    Demostration d=new Demostration() {
        void print()
        {
            System.out.println("Child Class");
        }
        };
    d.print();
    }
    void print()
    {
           System.out.println("Parent Class");
    }
}
```

Anonymous inner class

A class that have no name is known as anonymous inner class in java. It should be used if you have to override method of class or interface. Anonymous classes enable you to make your code more concise. They enable you to declare and instantiate a class at the same time. They are like local classes except that they do not have a name. Use them if you need to use a local class only once.

Java Anonymous inner class can be created by two ways:

- Class (may be abstract or concrete).
- Interface

References:

- http://programcall.com/8/csnet/oops-features-in-brief.aspx
- http://www.programmersnight.com/class-in-java/
- http://www.cpp-home.com/archives/206.html
- http://www.c4learn.com/java/java-assigning-object-reference/
- http://www.javatpoint.com/constructor
- http://www.sree9it.com/Java/constructors

Questions/Comments


THANK
YOU