Build COMPETENCY
across your TEAM

SYNERGETICS
GET IT RIGHT

Microsoft Partner
Gold Cloud Platform
Silver Learning

Classes and Objects

Google Cloud Platform
Windows Azure

amazon web services
APACHE kafka. A distributed streaming platform
Apache SOFTWARE FOUNDATION
APACHE Spark
Java
hadoop

Classes and Objects

Classes and Objects

Packages

Access Specifiers

Constructors - Default and Parameterized

this reference

using static keyword

Classes and Objects

- Classes and objects are the fundamental components of OOP's.
- Java is an Object-Oriented Language.
- Java supports the following fundamental OOP’s concepts
  - Polymorphism
  - Inheritance
  - Encapsulation
  - Abstraction
  - Classes
  - Objects
  - Instance
  - Method
  - Message Parsing

Classes and Objects

- **Object** – Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors – wagging the tail, barking, eating. An object is an instance of a class.

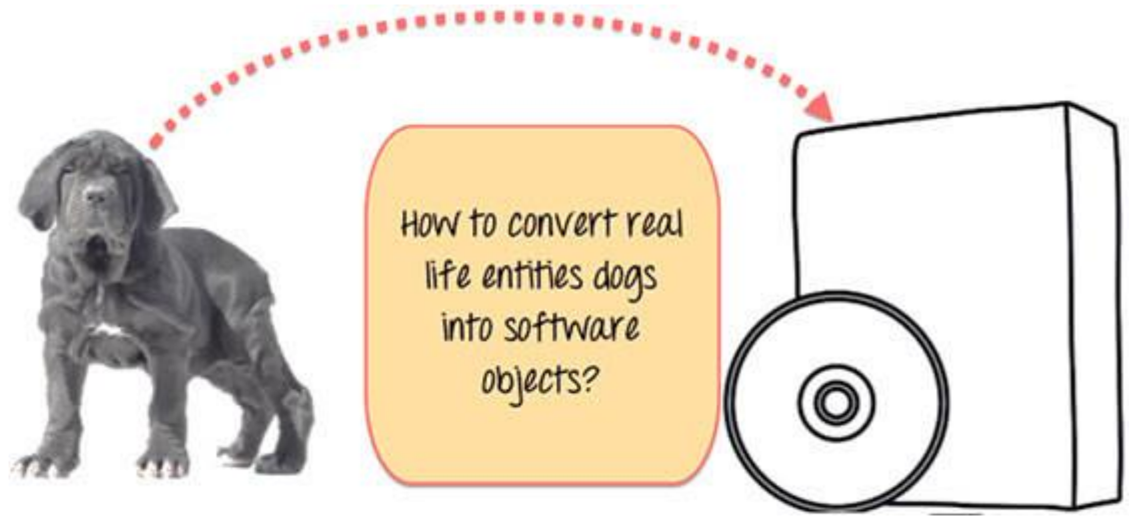
```
ClassName ReferenceVariable = new ClassName();
```

- **Class** – A class can be defined as a template/blueprint that describes the behavior/state that the object of its type support.

```
class <class_name>{
   field;
   method;
 }
```

Understand the concept of Java Classes and Objects with an example.

- Take an example of developing a pet management system, specially meant for dogs.
- We will need various information about the dogs like different breeds of the dogs, the age, size, etc.
- We need to model real-life beings, i.e., dogs into software entities.


How to convert real life entities dogs into software objects?

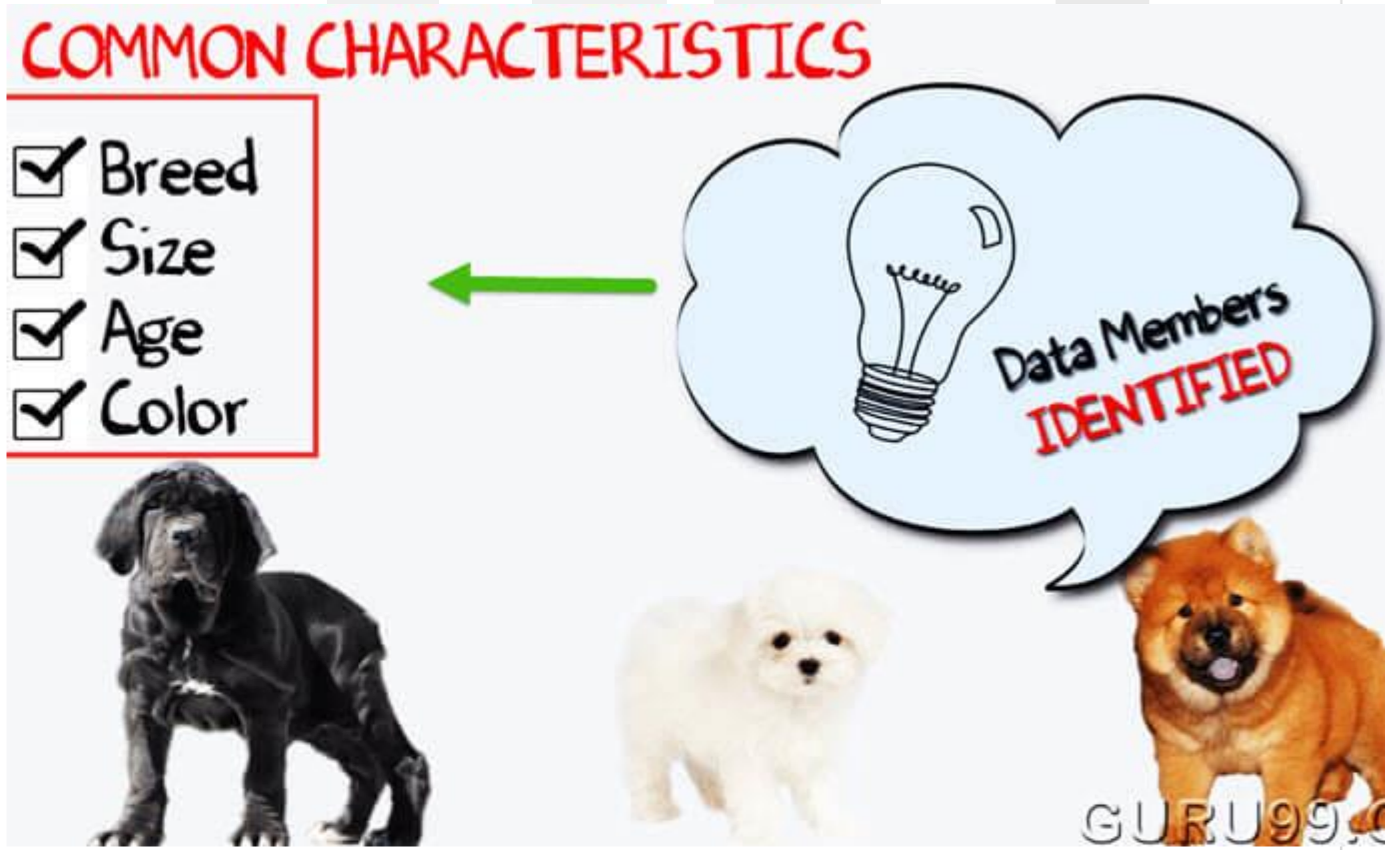
Understand the concept of Java Classes and Objects with an example

- You can see the picture of three different breeds of dogs below.

Stop here right now! List down the differences between them.

Understand the concept of Java Classes and Objects with an example

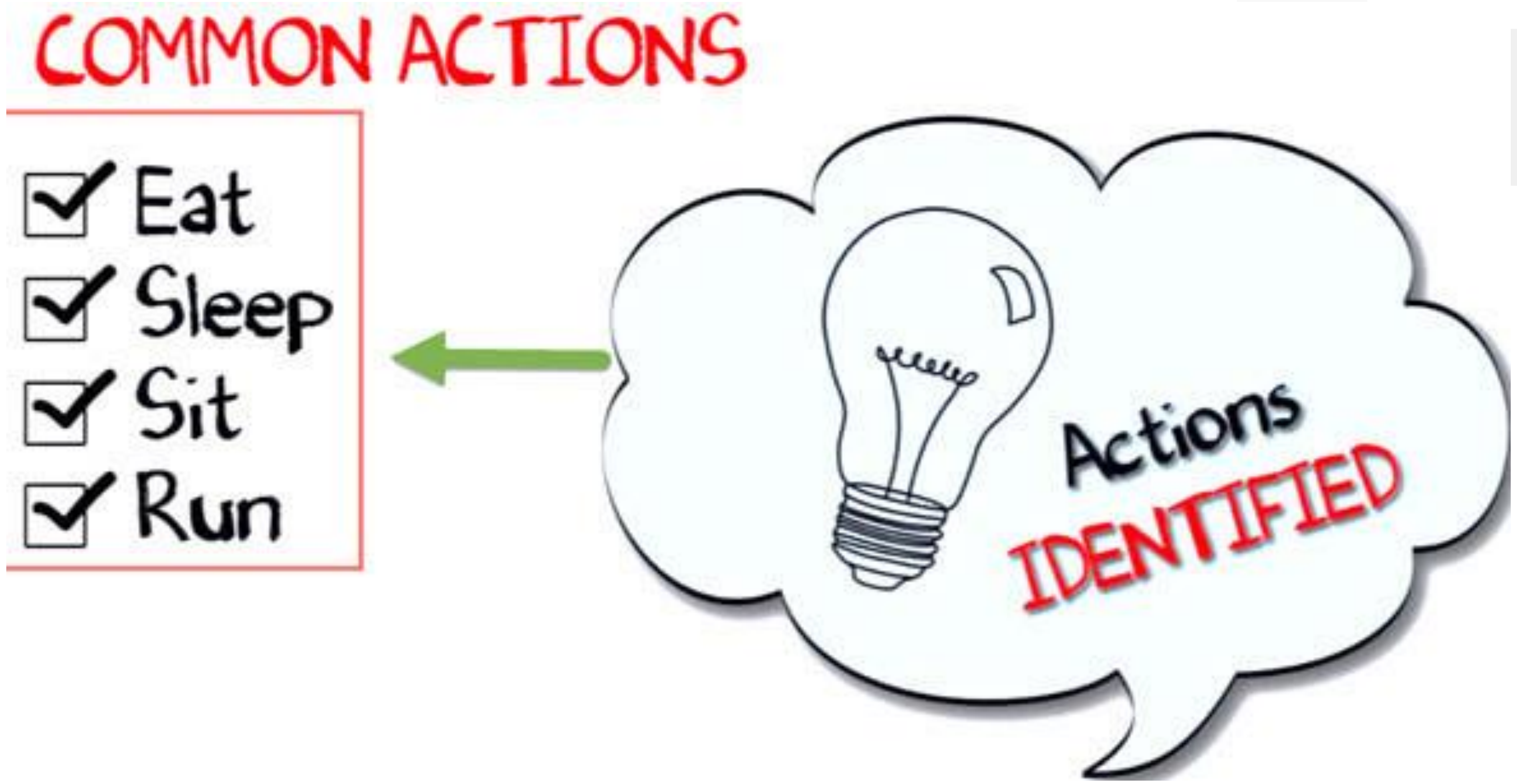
- Some of the differences you might have listed out maybe breed, age, size, color, etc.
- If you think for a minute, these differences are also some common characteristics shared by these dogs.
- These characteristics (breed, age, size, color) can form a data members for your object.



Understand the concept of Java Classes and Objects with an example

- Next, list out the common behaviors of these dogs like sleep, sit, eat, etc. So these will be the actions of our software objects.

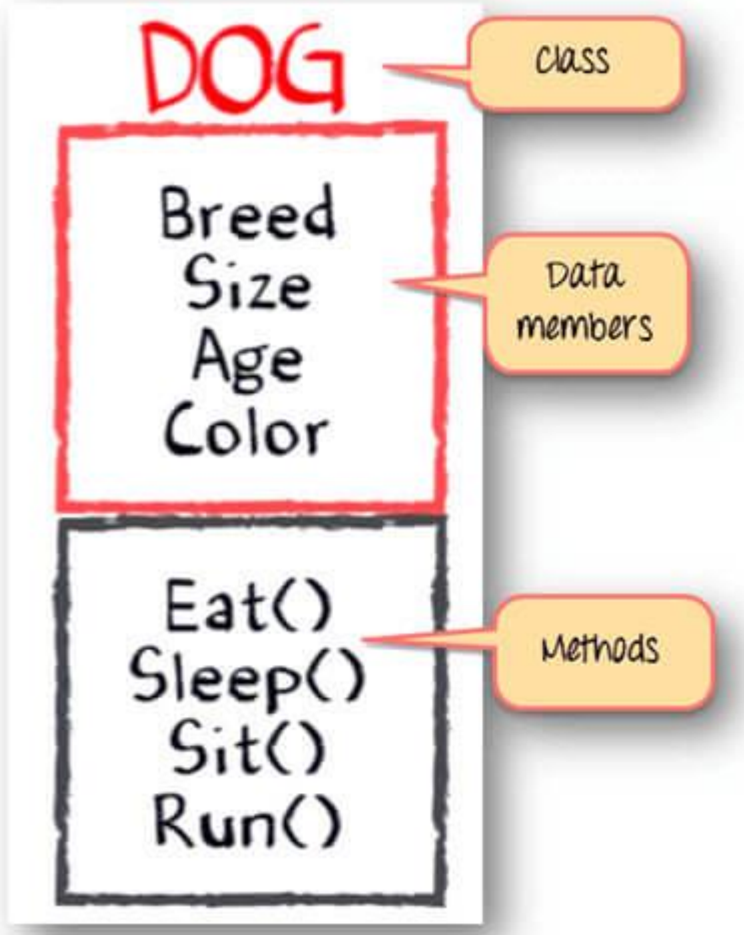
Understand the concept of Java Classes and Objects with an example



Understand the concept of Java Classes and Objects with an example

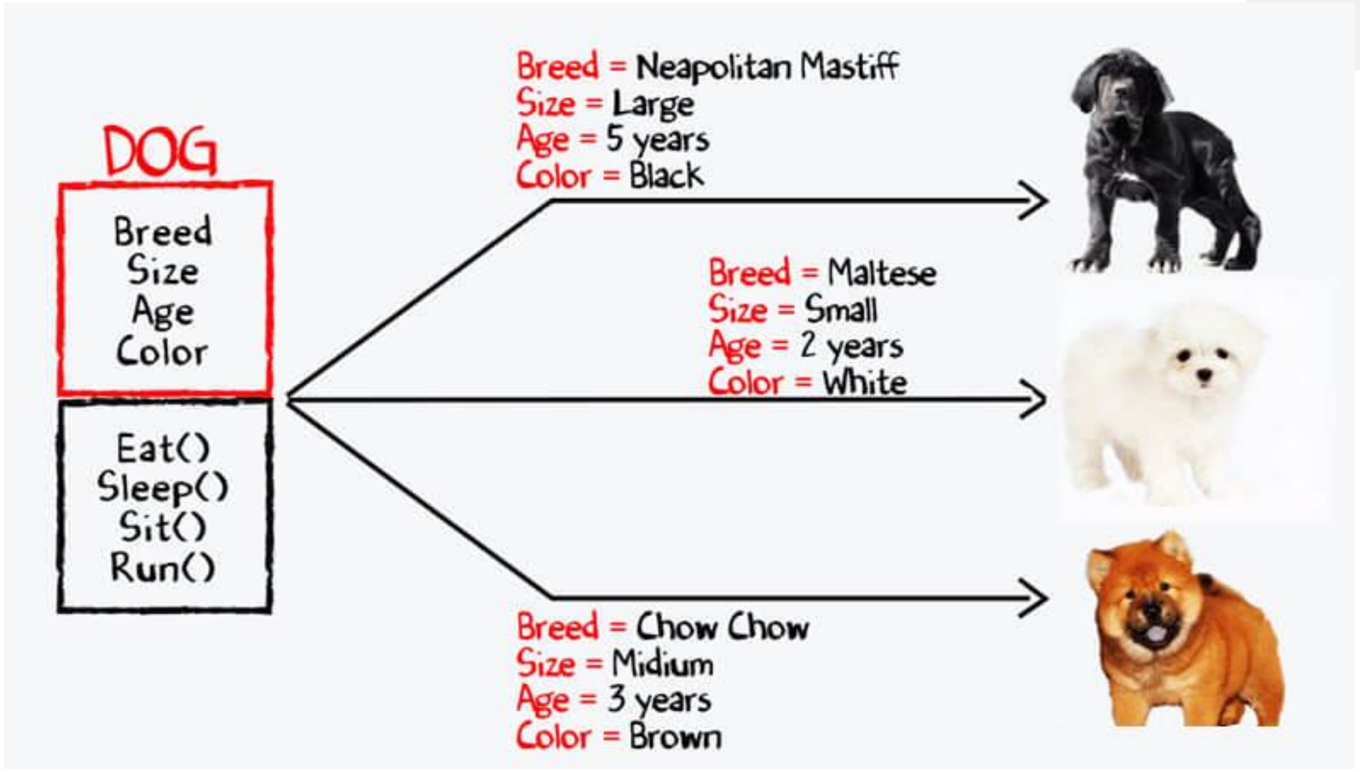
- So far we have defined following things

- **Class** - Dogs
- **Data members** or **objects**- size, age, color, breed, etc.
- **Methods**- eat, sleep, sit and run.



Understand the concept of Java Classes and Objects with an example

- Now, for different values of data members (breed size, age, and color) in Java class, you will get different dog objects



Important principles.

You can design any program using this OOPs approach.

However while creating a class, one must follow the following principles.

- **Single Responsibility Principle (SRP)-** A class should have only one reason to change
- **Open Closed Responsibility (OCP)-** It should be able to extend any classes without modifying it
- **Liskov Substitution Responsibility (LSR)-** Derived classes must be substitutable for their base classes
- **Dependency Inversion Principle (DIP)-** Depend on abstraction and not on concretions
- **Interface Segregation Principle (ISP)-** Prepare fine grained interfaces that are client specific.

Example

```
public class Dog {
    // Instance Variables
    String breed;
    String size;
    int age;
    String color;


    // method 1
    public String getInfo() {
        return ("Breed is: "+breed+" Size is:"+size+" Age is:"+age+" color is: "+color);
    }
```

Example

```
public static void main(String[] args) {
        Dog maltese = new Dog();
        maltese.breed="Maltese";
        maltese.size="Small";
        maltese.age=2;
        maltese.color="white";
        System.out.println(maltese.getInfo());
    }
}
```

Example – Main outside the class Dog

```
class Dog {
    // Instance Variables
    String breed;
    String size;
    int age;
    String color;



    // method 1
    public String getInfo() {
        return ("Breed is: "+breed+" Size is:"+size+" Age is:"+age+" color is: "+color);
    }
}
```

Example – Main outside the class Dog

- public class Execute{
-     public static void main(String[] args) {
-         Dog maltese = new Dog();
-         maltese.breed="Maltese";
-         maltese.size="Small";
-         maltese.age=2;
-         maltese.color="white";
-         System.out.println(maltese.getInfo());
-     }
- }

Packages in java

Packages in java

- A package as the name suggests is a pack(group) of classes, interfaces and other packages.
- In java we use packages to organize our classes and interfaces.
- We have two **types of packages in Java**: built-in packages and the packages we can create (also known as user defined package).
- In this guide we will learn what are packages, what are user-defined packages in java and how to use them.

Packages in java

- In java we have several built-in packages, for example when we need user input, we import a package like this:

```
import java.util.Scanner
```

Here:
→ **java** is a top level package
→ **util** is a sub package
→ and **Scanner** is a class which is present in the sub package **util**.

Advantages of using packages in Java

- **Reusability**
- **Better Organization**
- **Name Conflicts**

- **Reusability**: While developing a project in java, we often feel that there are few things that we are writing again and again in our code. Using packages, you can create such things in form of classes inside a package and whenever you need to perform that same task, just import that package and use the class.
- **Better Organization**: Again, in large java projects where we have several hundreds of classes, it is always required to group the similar types of classes in a meaningful package name so that you can organize your project better and when you need something you can quickly locate it and use it, which improves the efficiency.
- **Name Conflicts**: We can define two classes with the same name in different packages so to avoid name collision, we can use packages

Types of packages in Java

- User defined package: The package we create is called user-defined package
- Built-in package: The already defined package like java.io.*, java.lang.* etc are known as built-in packages.

Packages in java

```
package letmecalculate;

public class Calculator {
   public int add(int a, int b){
                        return a+b;
   }
   public static void main(String args[]){
                        Calculator obj = new Calculator();
                        System.out.println(obj.add(10, 20));
   }
}

=====================================================

import letmecalculate.Calculator;
public class Demo{
   public static void main(String args[]){
                        Calculator obj = new Calculator();
                        System.out.println(obj.add(100, 200));
   }
}
```

To use the class Calculator, we have imported the package letmecalculate.

Packages in java

- **Creating a class inside package while importing another package**

```
//Declaring a package
package anotherpackage;

//importing a package
import letmecalculate.Calculator;

public class Example{
  public static void main(String args[]){
                Calculator obj = new Calculator();
                System.out.println(obj.add(100, 200));
  }
}
```

So the order in this case should be:

→ package declaration

→ package import

- **Using fully qualified name while importing a class**
- You can use fully qualified name to avoid the import statement.

```
package letmecalculate;
public class Calculator {
   public int add(int a, int b){
            return a+b;
   }
   public static void main(String args[]){
            Calculator obj = new Calculator();
            System.out.println(obj.add(10, 20));
   }
}
```

```
//Declaring a package
package anotherpackage;
public class Example{
   public static void main(String args[]){
        //Using fully qualified name instead of import
            letmecalculate.Calculator obj =
                  new letmecalculate.Calculator();
            System.out.println(obj.add(100, 200));
   }
}
```

Access Specifiers

Access Modifiers in Java

As the name suggests access modifiers in Java helps to restrict the scope of a class, constructor , variable , method or data member. There are four types of access modifiers available in java:

- Default – No keyword required
- Private
- Protected
- Public

	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Access Modifiers in Java

- **Default**: When no access modifier is specified for a class , method or data member – It is said to be having the **default** access modifier by default.

- **Private**: The private access modifier is specified using the keyword **private**.
  - The methods or data members declared as private are accessible only **within the class** in which they are declared.
  - Any other **class of same package will not be able to access** these members.
  - Classes or interface can not be declared as private.

- **protected**: The protected access modifier is specified using the keyword **protected**.
  - The methods or data members declared as protected are **accessible within same package or sub classes in different package.**

- **public**: The public access modifier is specified using the keyword **public**.
  - The public access modifier has the **widest scope** among all other access modifiers.
  - Classes, methods or data members which are declared as public are **accessible from every where** in the program. There is no restriction on the scope of a public data members.

Important Points:

- If other programmers use your class, try to use the most restrictive access level that makes sense for a particular member. Use private unless you have a good reason not to.
- Avoid public fields except for constants.

Constructors - Default and Parameterized

What is Constructor in Java?

- A **constructor** is a special method that is used to initialize a newly created object and is called just after the memory is allocated for the object.
- It can be used to initialize the objects to desired values or default values at the time of object creation.
- It is not mandatory for the coder to write a constructor for a class.
- If no user-defined constructor is provided for a class, compiler initializes member variables to its default values.
    - numeric data types are set to 0
    - char data types are set to null character('\0')
    - reference variables are set to null

Rules for creating a Java Constructor

- It has the **same name** as the class
- It should not return a value not even ***void***

```
class Square
{
        int side;

        public Square()
        {
                side = 0;
        }
}
```

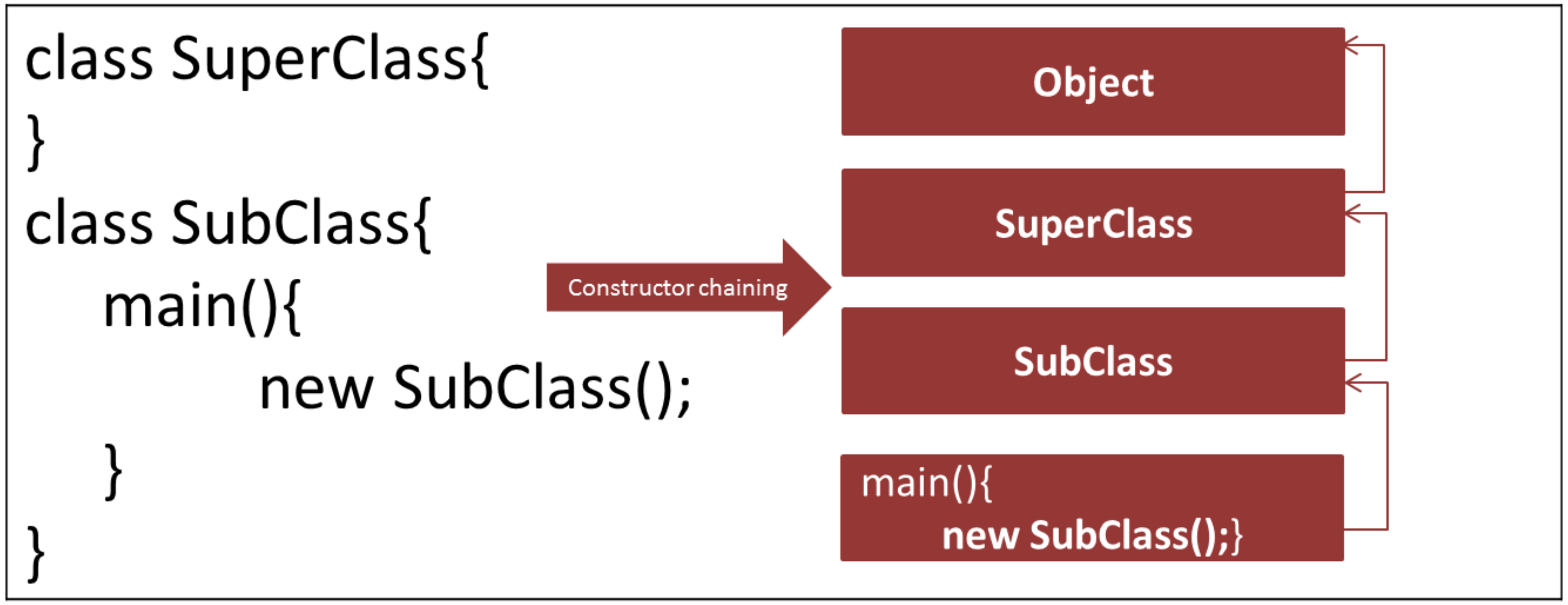
Constructor Overloading

- Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter list.
- The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

- Circle c = new Circle();
- Circle c = new Circle(12);

Constructor Chaining

- Consider a scenario where a base class is extended by a child.
- Whenever an object of the child class is created, the constructor of the parent class is invoked first. This is called **Constructor chaining.**



```
class SuperClass{
}
class SubClass{
    main(){
        new SubClass();
    }
}
```

this reference

The *this* Keyword

- ***This*** is a keyword in Java. this keyword in java can be used inside the M*ethod* or *constructor* of Class.
- It(***this)*** works as a reference to the current Object whose Method or constructor is being invoked.
- This keyword can be used to refer to any member of the current object from within an instance Method or a constructor.

- ***this*** keyword can be very useful in the handling of Variable Hiding. We can not create two instance/local variables with the same name. However, it is legal to create one instance variable & one local variable or Method parameter with the same name. In this scenario, the local variable will hide the instance variable this is called Variable Hiding.

The *this* Keyword

```
class Employee{
        private String name;
        private int id;

        public Employee( String name, int id ){
                this.name = name;
                this.id = id;
}
```

//without the this reference stating the left side of the assignment is a class variable the compiler would this is a local variable declared in the constructor and the value would never reach the instance variable

Using Static Keyword

SYNERGETICS GET IT RIGHT

Microsoft Partner
Gold Cloud Platform
Silver Learning

Static keyword in java

- The static keyword is used in java mainly for memory management. It is used with variables, methods, blocks and nested class. It is a keyword that are used for share the same variable or method of a given class. This is used for a constant variable or a method that is the same for every instance of a class. The main method of a class is generally labeled static.

- No object needs to be created to use static variable or call static methods, just put the class name before the static variable or method to use them. Static method can not call non-static method.

Static keyword in java

In java language static keyword can be used for following

- variable (also known as class variable)
- method (also known as class method)
- block
- nested class

Static keyword in java

- **Static variable**

If any variable we declared as static is known as static variable.

- Static variable is used for fulfill the common requirement. For Example company name of employees,college name of students etc. Name of the college is common for all students.

- The static variable allocate memory only once in class area at the time of class loading.

Static keyword in java

- **Advantage of static variable**
  - Using static variable we make our program memory efficient (i.e it saves memory).

When and why we use static variable

Suppose we want to store record of all employee of any company, in this case employee id is unique for every employee but company name is common for all. When we create a static variable as a company name then only once memory is allocated otherwise it allocate a memory space each time for every employee.

Static keyword in java

Syntax for declare static variable:

```
public static variableName;
```

Syntax for declare static method:

```
public static  void methodName()
{
.......
.......
}
```

Syntax for access static methods and static variable

```
className.variableName=10;
className.methodName();
```

Static keyword in java

- **Difference between static and final keyword**

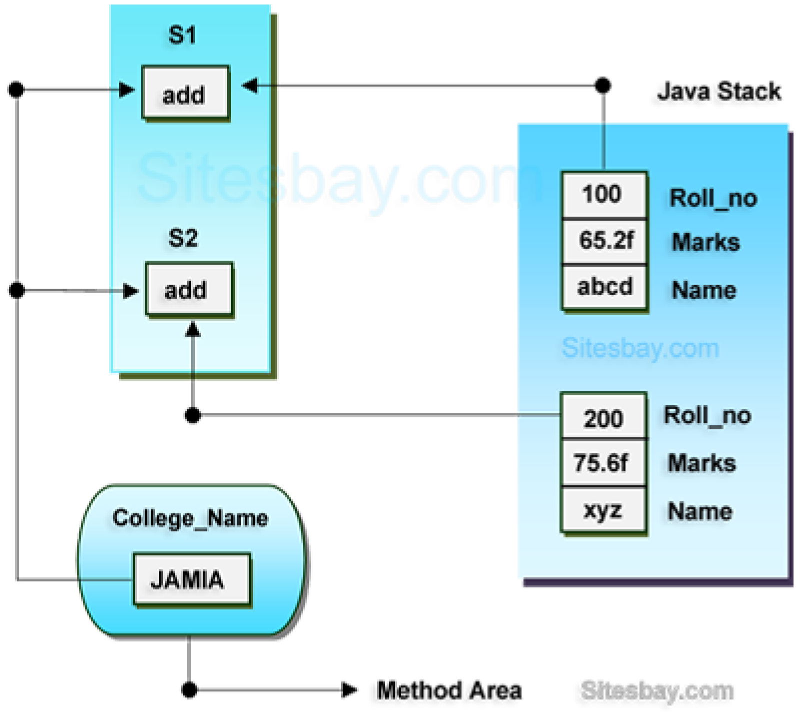
- **static** keyword always fixed the memory that means that will be located only once in the program where as **final** keyword always fixed the value that means it makes variable values constant.
- **Note:** As for as real time statement there concern every final variable should be declared the static but there is no compulsion that every static variable declared as final.

Example

```
class Student
{
int roll_no;
String name;
static String College_Name="ITM";
}
class StaticDemo
{
public static void main(String args[])
{
Student s1=new Student();
s1.roll_no=100;
s1.name="abcd";
System.out.println(s1.roll_no);
System.out.println(s1.name);
System.out.println(Student.College_Name);
Student  s2=new  Student();
s2.roll_no=200;
s2.name="zyx";
System.out.println(s2.roll_no);
System.out.println(s2.name);
System.out.println(Student.College_Name);
}
}
```

Example Continued

- In the above example College_Name variable is commonly sharable by both S1 and S2 objects.
- In the image static data variable are store in method area and non static variable is store in java stack.



Static keyword in java

- **Why Main Method Declared Static ?**

- Because object is not required to call static method if main() is non-static method, then jvm create object first then call main() method due to that face the problem of extra memory allocation.

SYNERGETICS GET IT RIGHT | Microsoft Partner Gold Cloud Platform Silver Learning

Q & A

Contact: amitmahadik@synergetics-india.com

SYNERGETICS GET IT RIGHT

Microsoft Partner
Gold Cloud Platform
Silver Learning

Thank You