Object Oriented Concepts

OOP

- A principle of design and development of programs using modular approach
- A computer programming model that organizes software design around **data**, or **objects**, rather than functions and logic.
- well-suited for programs that are large, complex and actively updated or maintained.

Object

- Objects are basic building blocks for designing programs
- Has unique attributes and behavior
- A collection of data members and associated member functions that manipulate that data members
- An object is active, not passive; it does things
- An object is responsible for its own data
- But: it can expose that data to other objects

Class

- Objects can be made with the help of a class.
- A class is a collection of objects that have identical properties, common behavior and shared relationship
- Hold both data and methods

Classes and Objects

Class
<CAR>

Example of a class

```
class Employee {
    // Fields
    private String name;      //Can get but not change
    private double salary;   // Cannot get or set
    // Constructor
    Employee(String n, double s) {
        name = n; salary = s;
    }
    // Methods
    void pay () {
        System.out.println("Pay to the order of " +
                           name + " $" + salary);
    }
    public String getName() { return name; } // getter
}
```

Concept: Objects must be created

- int n; does two things:
  - It declares that n is an integer variable
  - It allocates space to hold a value for n
  - For a primitive, this is all that is needed
- Employee secretary; also does two things
  - It declares that secretary is type Employee
  - It allocates space to hold a *reference* to an Employee
  - For an object, this is ***not*** all that is needed
- secretary = new Employee ( );
  - This allocate space to hold a *value* for the Employee
  - Until you do this, the Employee is null

Notation: How to declare and create objects

```
Employee secretary;  // declares secretary
secretary = new Employee (); // allocates space
Employee secretary = new Employee(); // does both
```

- But the secretary is still "blank" (null)

```
secretary.name = "Adele";  // dot notation
secretary.birthday ();  // sends a message
```

Notation: How to reference a field or method

- Inside a class, no dots are necessary

```
class Person { … age = age + 1; …}
```

- Outside a class, you need to say which object you are talking to

```
if (john.age < 75) john.birthday ();
```

- If you don't have an object, you cannot use its fields or methods!

Concept: this object

- Inside a class, no dots are necessary, because
  - you are working on this object
- If you wish, you can make it explicit:

  class Person { … this.age = this.age + 1; …}
- this is like an extra parameter to the method
- You usually don't need to use this

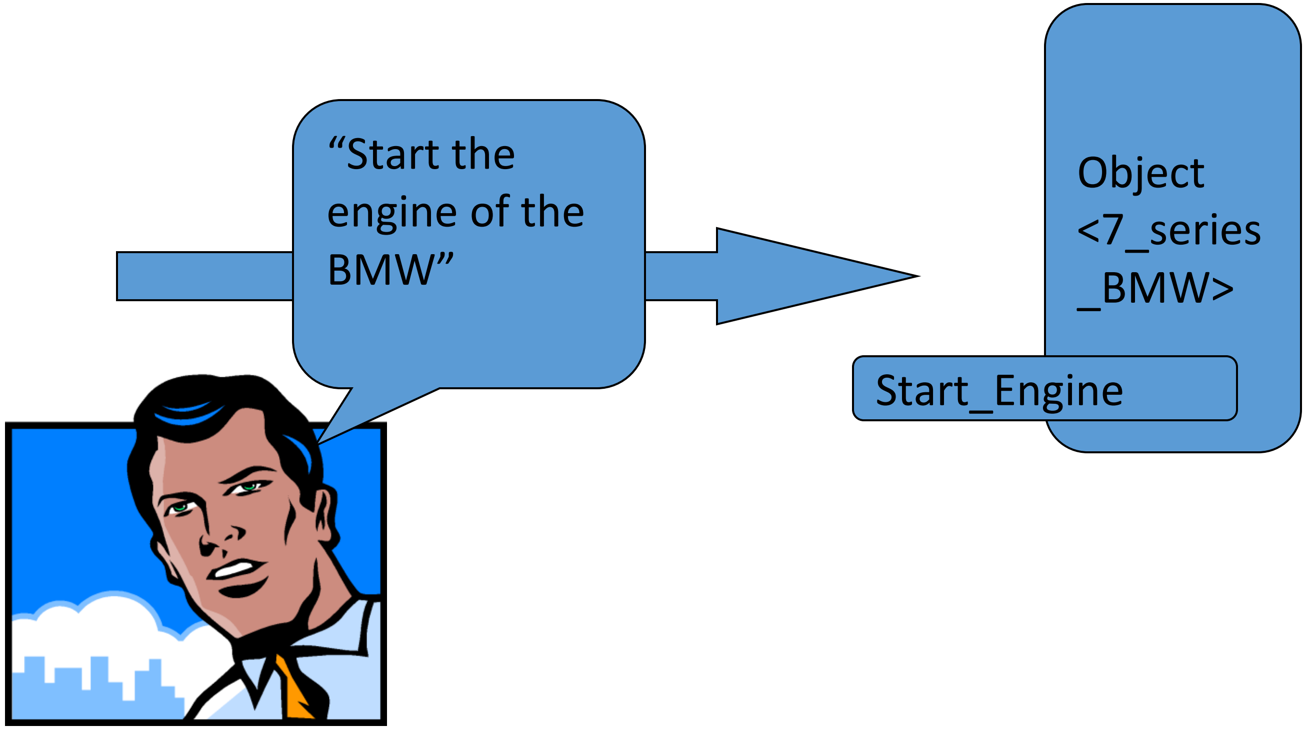
Classes and Objects

Object <7_series_BMW>

Object <VW_Beetle>

Object <Ford_Mustang>

Messages to Objects

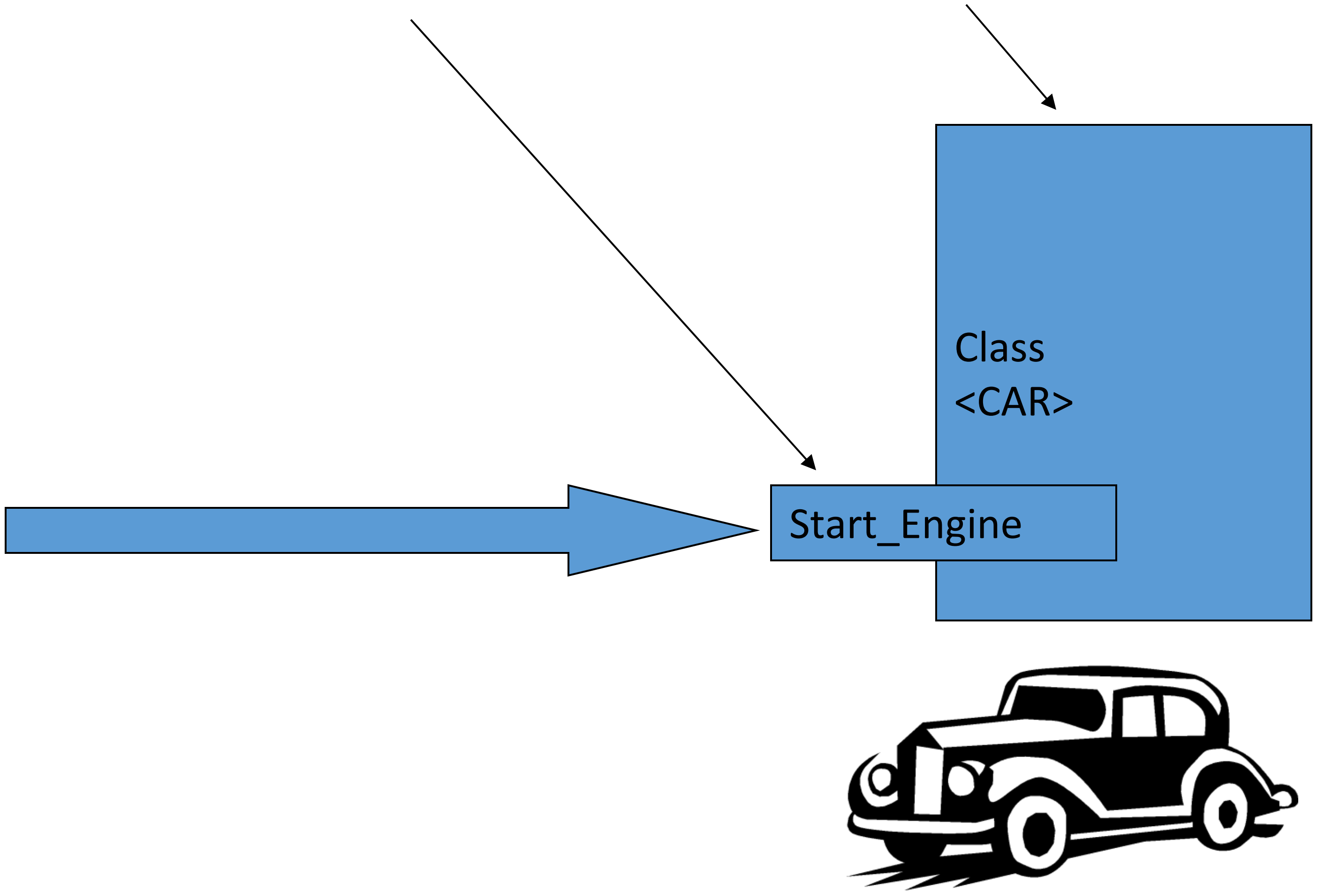


"Start the engine of the BMW"

Object <7_series _BMW>

Start_Engine

Method of a Class



Class
<CAR>

Start_Engine

Types of Methods

- There are 4 basic types of methods:
  - Modifier (sometimes called a mutator)
    - Changes the value associated with an attribute of the object
    - E.g. A method like ***Change_Car_Color***
  - Accessor
    - Returns the value associated with an attribute of the object
    - E.g. A method like ***Price_of_Car***
  - Constructor
    - Called once when the object is created (before any other method will be invoked)
    - E.g. ***Car(Mustang)***
  - Destructor
    - Called when the object is destroyed
    - E.g***.~Car( )***

Constructors

- Constructors are used to initialize the object’s state.
- Like methods, a constructor also contains collection of statements(i.e. instructions) that are executed at time of Object creation.
- 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.
- Constructor is invoked at the time of object or instance creation.

Regulation

- A constructor doesn’t have a return types
- The name of the constructor must be the same as the name of the class.
- Unlike methods, constructors are not considered members of a class.
- A constructor is called automatically when a new instance of an object is created.

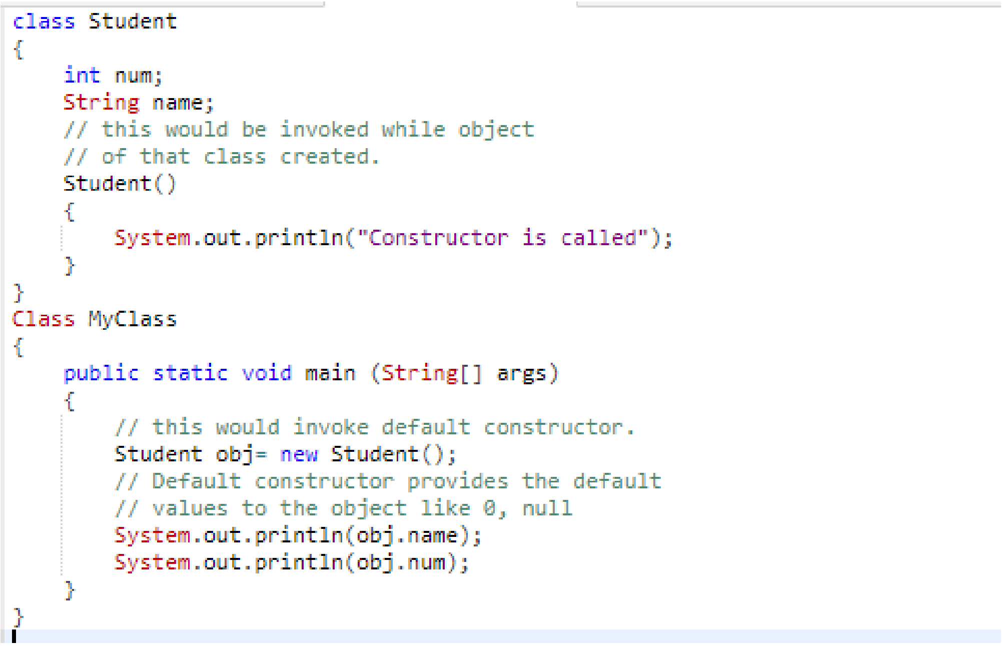
Types of Constructor

- There are two types of constructors in java
  - Default constructor (no-argument constructor)
  - Parameterized constructor

Default Constructor

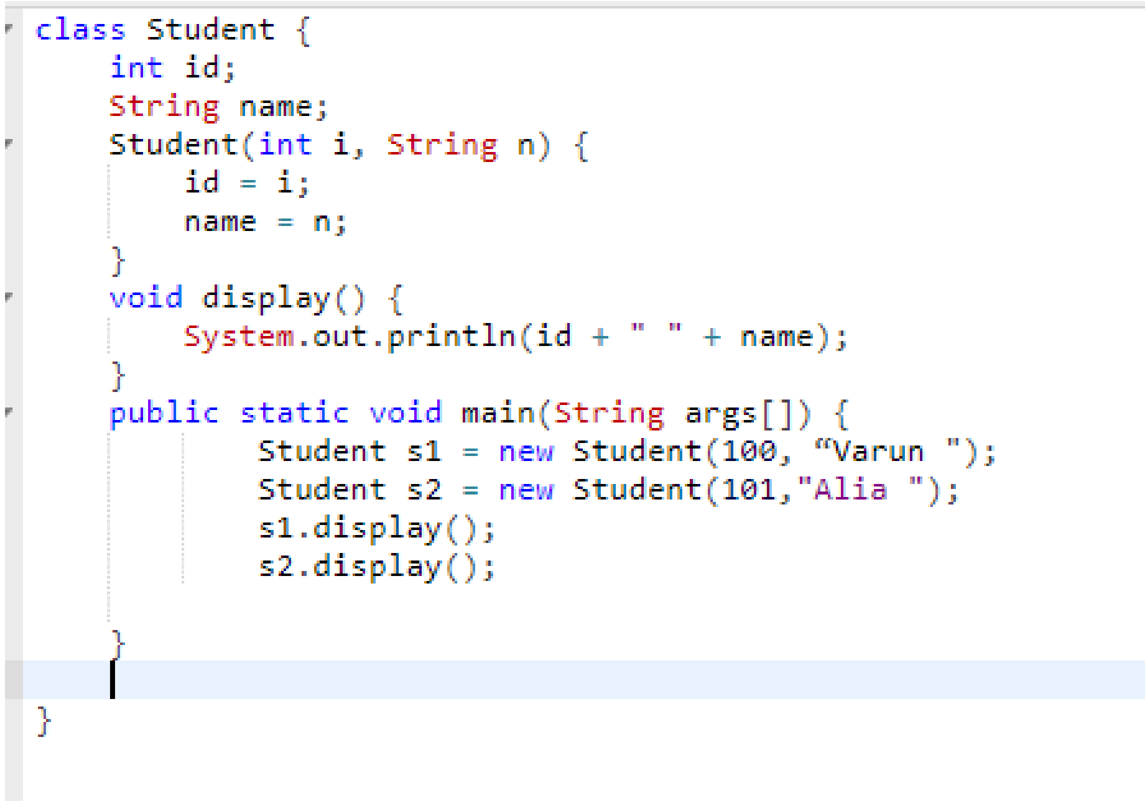
- A constructor is called "Default Constructor" when it doesn't have any parameter.
- If we don’t define a constructor in a class, then compiler creates default constructor(with no arguments) for the class.
- If we write a constructor with arguments or no-argument then compiler does not create default constructor.
- Default constructor provides the default values to the object like 0, null etc. depending on the type.

Default Constructor Example



```
class Student
{
    int num;
    String name;
    // this would be invoked while object
    // of that class created.
    Student()
    {
        System.out.println("Constructor is called");
    }
}
Class MyClass
{
    public static void main (String[] args)
    {
        // this would invoke default constructor.
        Student obj= new Student();
        // Default constructor provides the default
        // values to the object like 0, null
        System.out.println(obj.name);
        System.out.println(obj.num);
    }
}
```

Parameterized Constructor Example

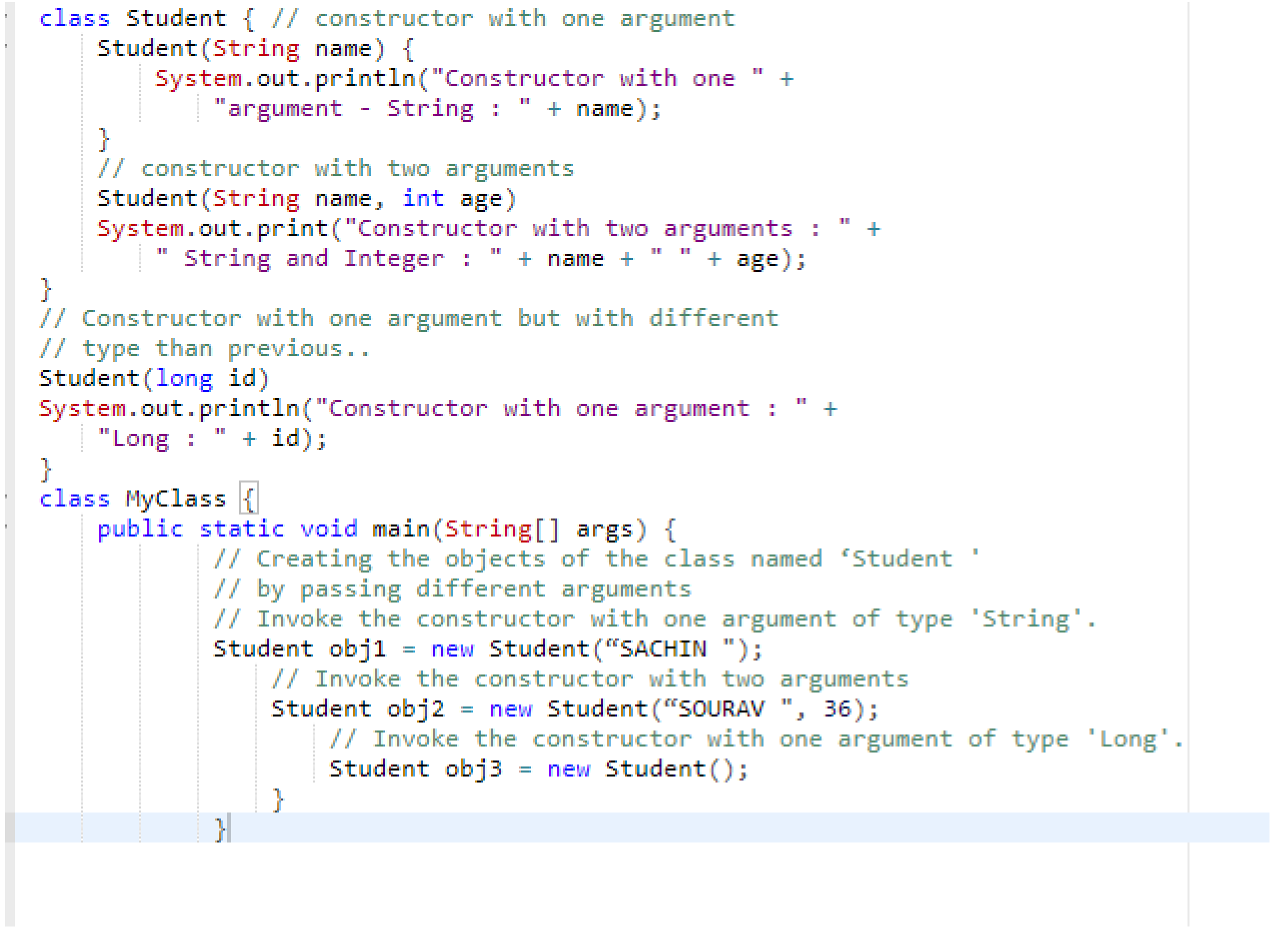


```
class Student {
    int id;
    String name;
    Student(int i, String n) {
        id = i;
        name = n;
    }
    void display() {
        System.out.println(id + " " + name);
    }
    public static void main(String args[]) {
            Student s1 = new Student(100, “Varun ");
            Student s2 = new Student(101,"Alia ");
            s1.display();
            s2.display();

    }

}
```

Constructor Overloading Example

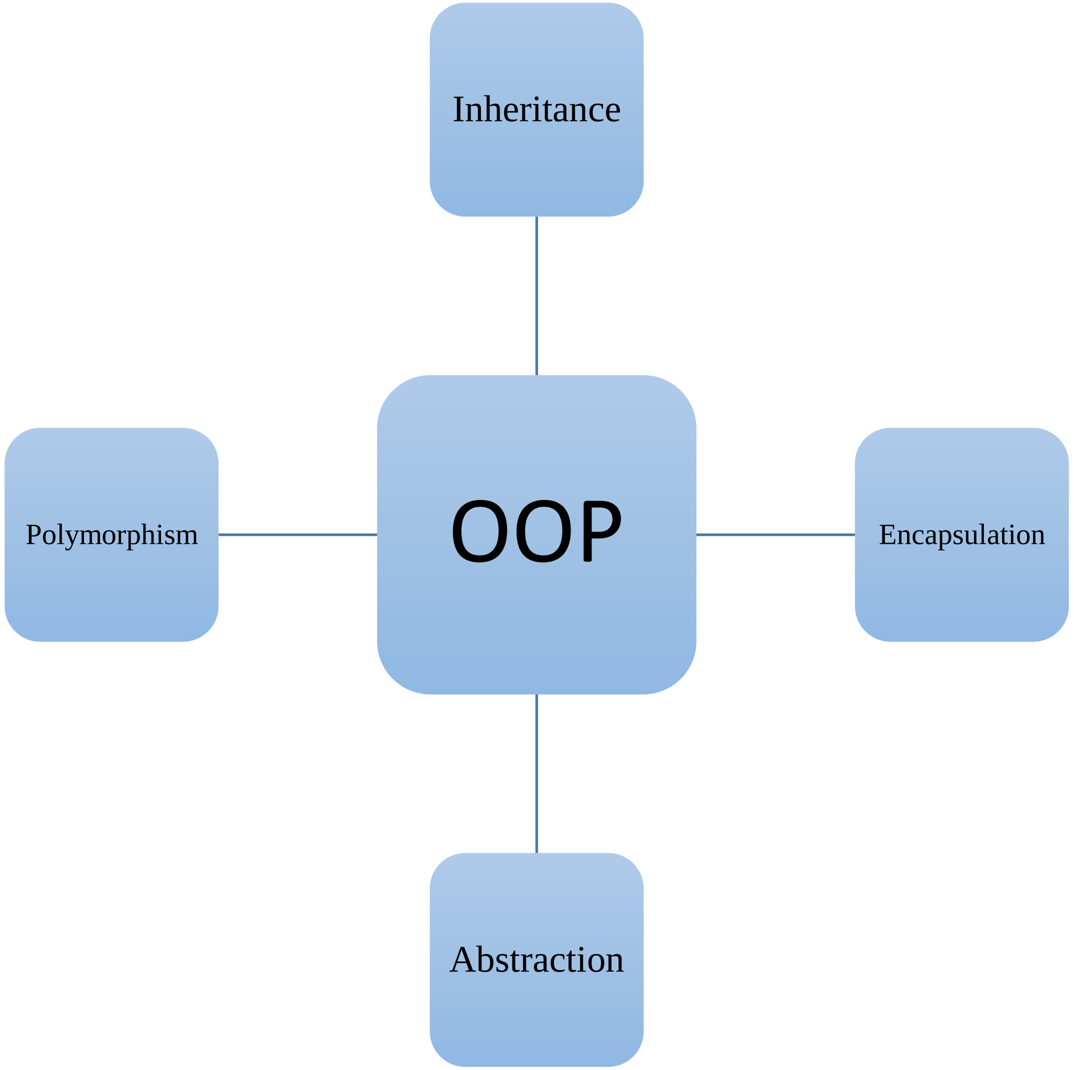


```
class Student { // constructor with one argument
    Student(String name) {
        System.out.println("Constructor with one " +
            "argument - String : " + name);
    }
    // constructor with two arguments
    Student(String name, int age)
    System.out.print("Constructor with two arguments : " +
        " String and Integer : " + name + " " + age);
}
// Constructor with one argument but with different
// type than previous..
Student(long id)
System.out.println("Constructor with one argument : " +
    "Long : " + id);
}
class MyClass {
    public static void main(String[] args) {
            // Creating the objects of the class named ‘Student '
            // by passing different arguments
            // Invoke the constructor with one argument of type 'String'.
            Student obj1 = new Student(“SACHIN ");
                // Invoke the constructor with two arguments
                Student obj2 = new Student(“SOURAV ", 36);
                    // Invoke the constructor with one argument of type 'Long'.
                    Student obj3 = new Student();
                }
            }
```

Access

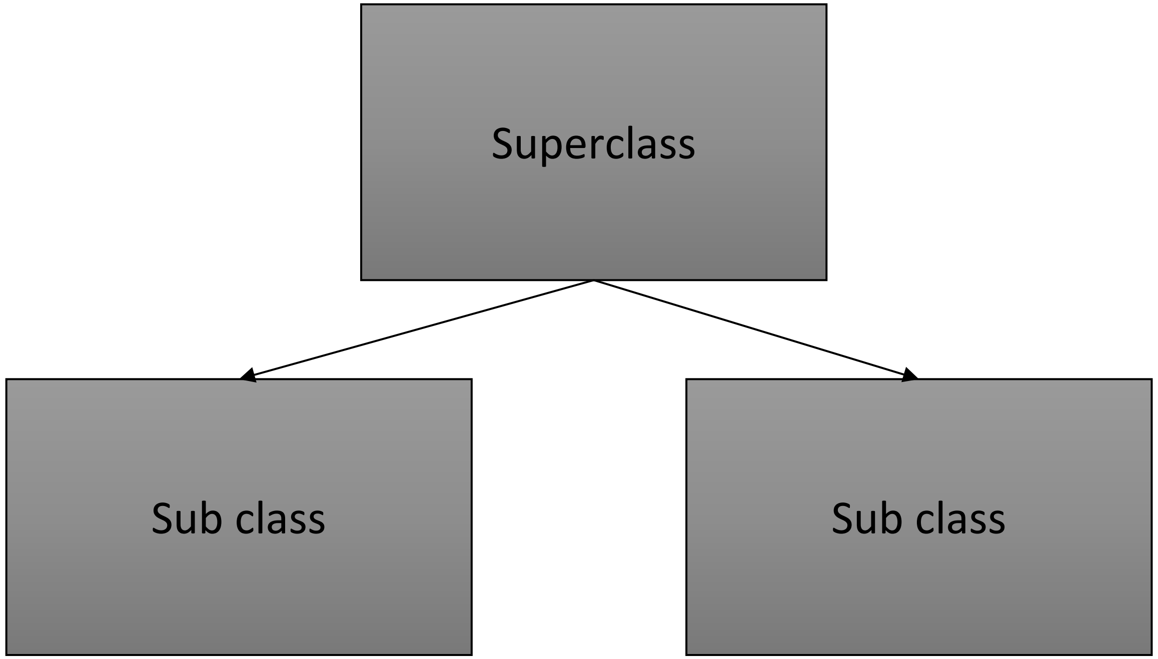
- Most classes provide three levels of access to their members (state and behavior):
  - Public
    - The part of the class of the class that is visible to all clients of the class
  - Protected
    - The part of the class that is only visible to subclasses of the class
  - Private
    - A part of the class that is not visible to any other classes

Components of OOP



Inheritance

Inheritance is a relationship where one class shares the structure or behavior defined in one class (single inheritance) or more (multiple inheritance)



Benefits of Inheritance

- One view of inheritance is that it provides a way to specify some properties/behaviors that all subclasses *must* exhibit
- Inheritance can be used to re-use code
- Inheritance also provides the ability to generalize
  - A method can be written to work with the super-class but subclasses can be passed as arguments

Example: Assignment of subclasses

```
class Dog { … }
class Poodle extends Dog { … }
Dog myDog;
Dog rover = new Dog ();
Poodle yourPoodle;
Poodle fifi = new Poodle ();

myDog = rover;                      // ok
yourPoodle = fifi;                  // ok
myDog = fifi;                       //ok
yourPoodle = rover;                 // illegal
yourPoodle = (Poodle) rover;        //runtime check
```

Example of inheritance

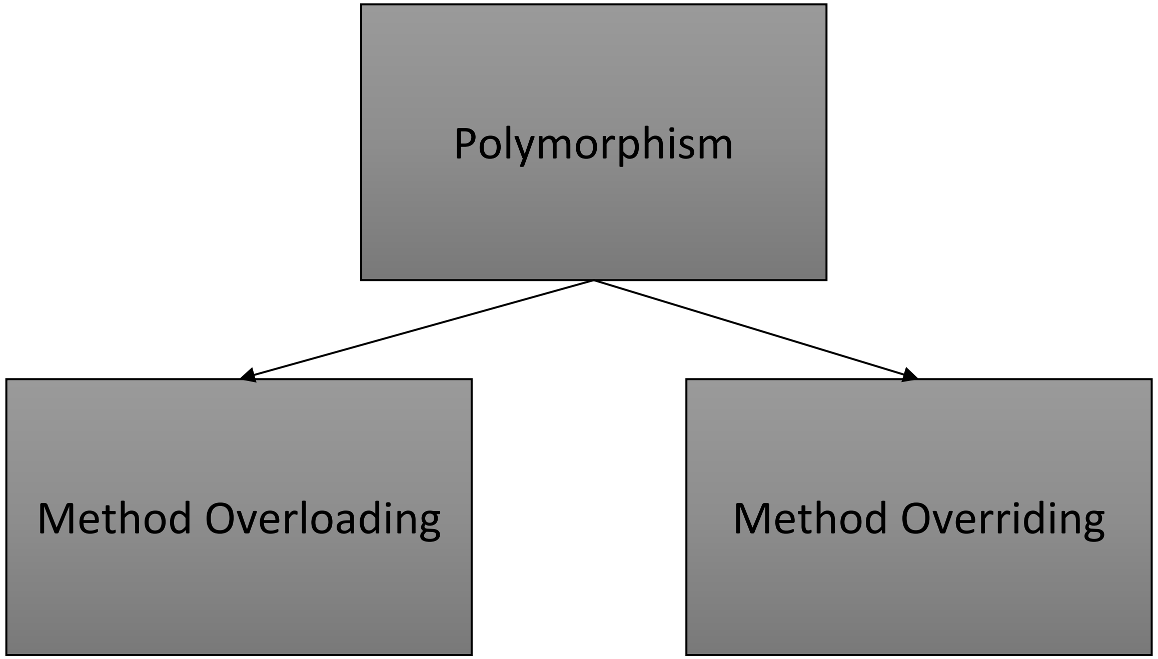
```
class Person {
  String name;
  int age;
  void birthday () {
    age = age + 1;
  }
}
```

```
class Employee
      extends Person {
  double salary;
  void pay () { ...}
}
```

Every Employee has name and age fields and birthday method *as well as* a salary field and a pay method.

Polymorphism

Ability of a function to take multiple forms



Polymorphism

Method Overloading

Method Overriding

Methods can be overridden

```
class Bird extends Animal {
  void fly (String destination) {
    location = destination;
  }
}

class Penguin extends Bird {
  void fly (String whatever) { }
}
```

- So birds can fly. Except penguins.

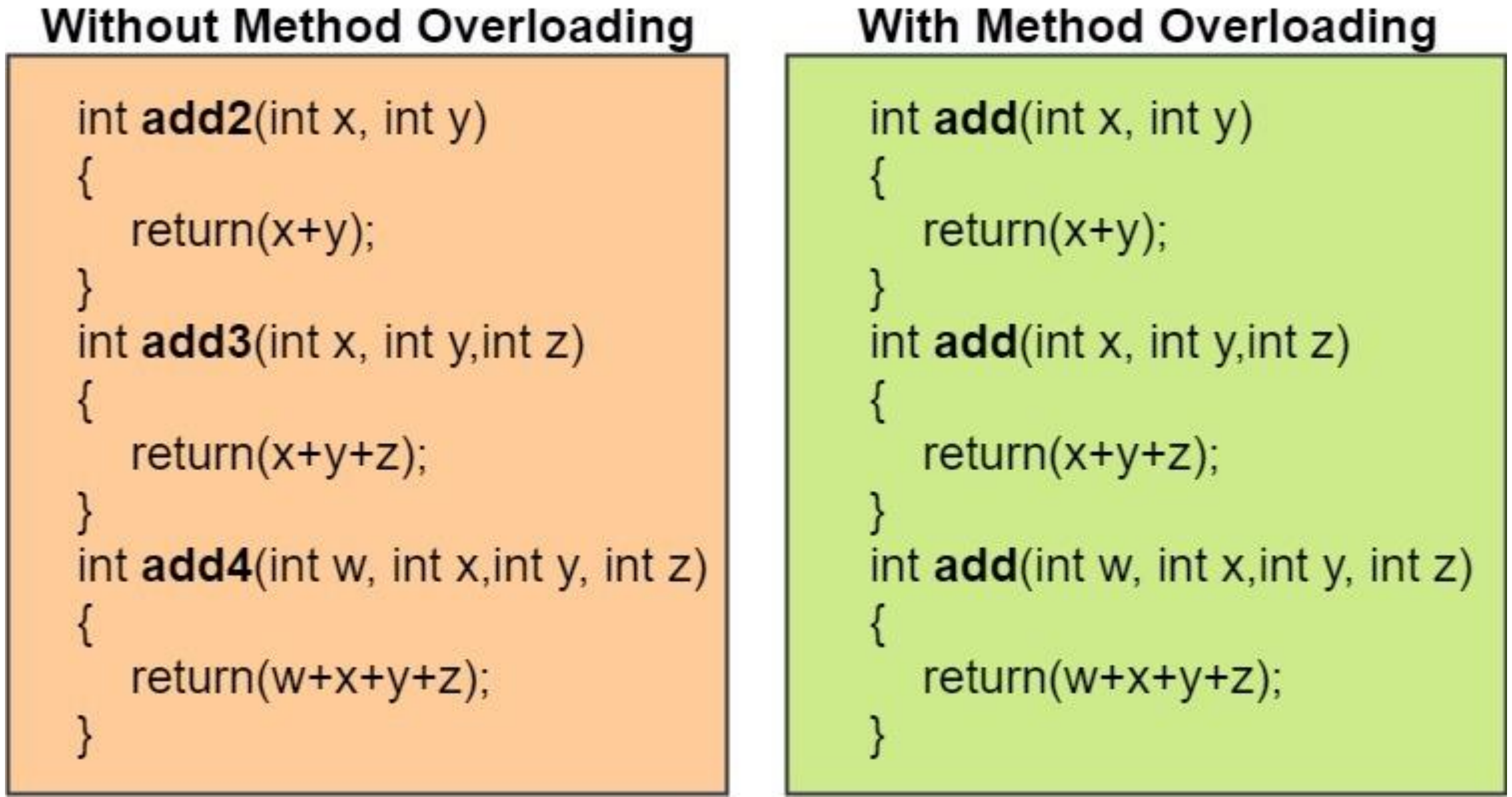
How to use overridden methods

```
class FamilyMember extends Person {
  void birthday () { // override birthday() in Person
    super.birthday ();  // call overridden method
    givePresent ();     // and add your new stuff
  }
}
```

Overloading

Allows object to have different meaning depending upon context

- **Operator overloading:** when an existing operator operates on new data type is called operator overloading
- **Function overloading:** means two or more function have same name, but differ in the number of arguments or data type of arguments.



Without Method Overloading

```
int add2(int x, int y)
{
    return(x+y);
}
int add3(int x, int y,int z)
{
    return(x+y+z);
}
int add4(int w, int x,int y, int z)
{
    return(w+x+y+z);
}
```

With Method Overloading

```
int add(int x, int y)
{
    return(x+y);
}
int add(int x, int y,int z)
{
    return(x+y+z);
}
int add(int w, int x,int y, int z)
{
    return(w+x+y+z);
}
```

Encapsulation

```
class Employee extends Person {
   private double salary;
   private boolean male;
   public void setSalary (double newSalary) {
      salary = newSalary;
   }
   public double getSalary () { return salary; }
   public boolean isMale() { return male; }
}
```

- This way the object maintains control
- Setters and getters have conventional names: set***DataName***, get***DataName***, is***DataName*** (booleans only)

Abstraction and Interface