OOP

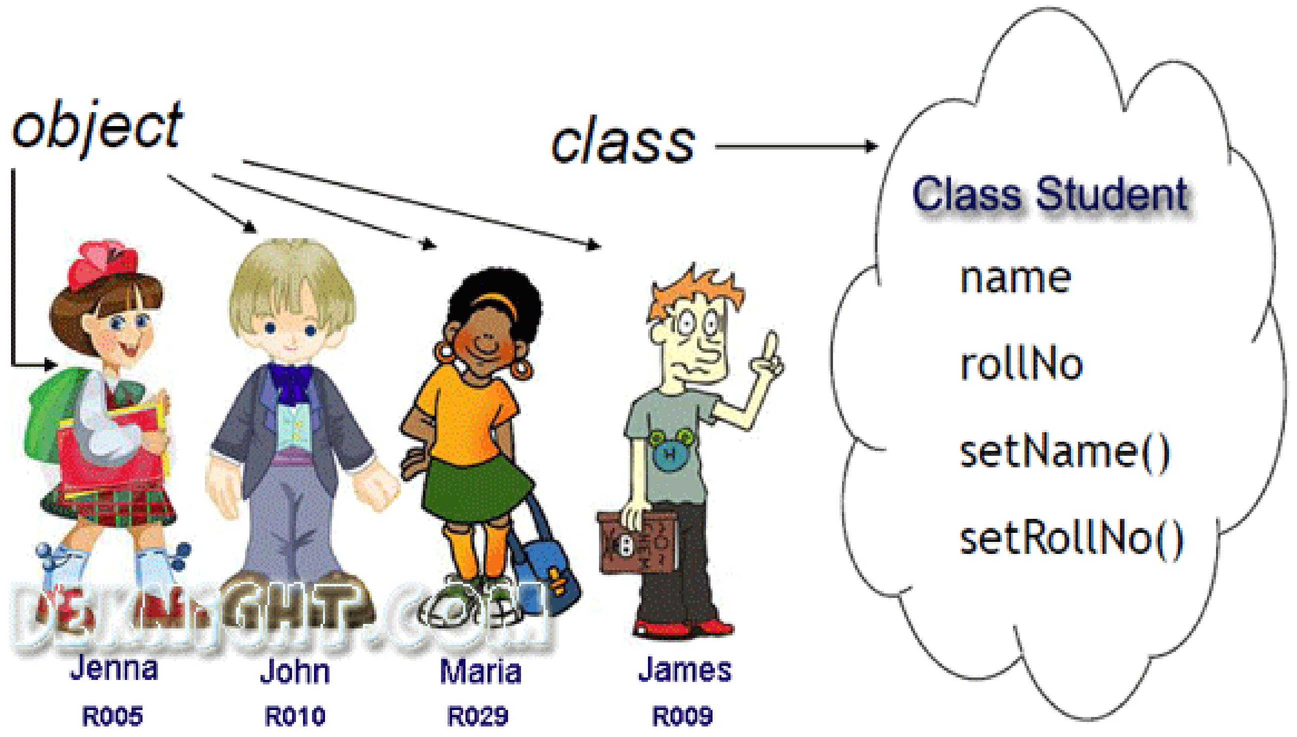
Content

- Class & Object
- Encapsulation
- Inheritance
- Polymorphism
- Abstraction
- Interfaces

OOP

From Class to Object

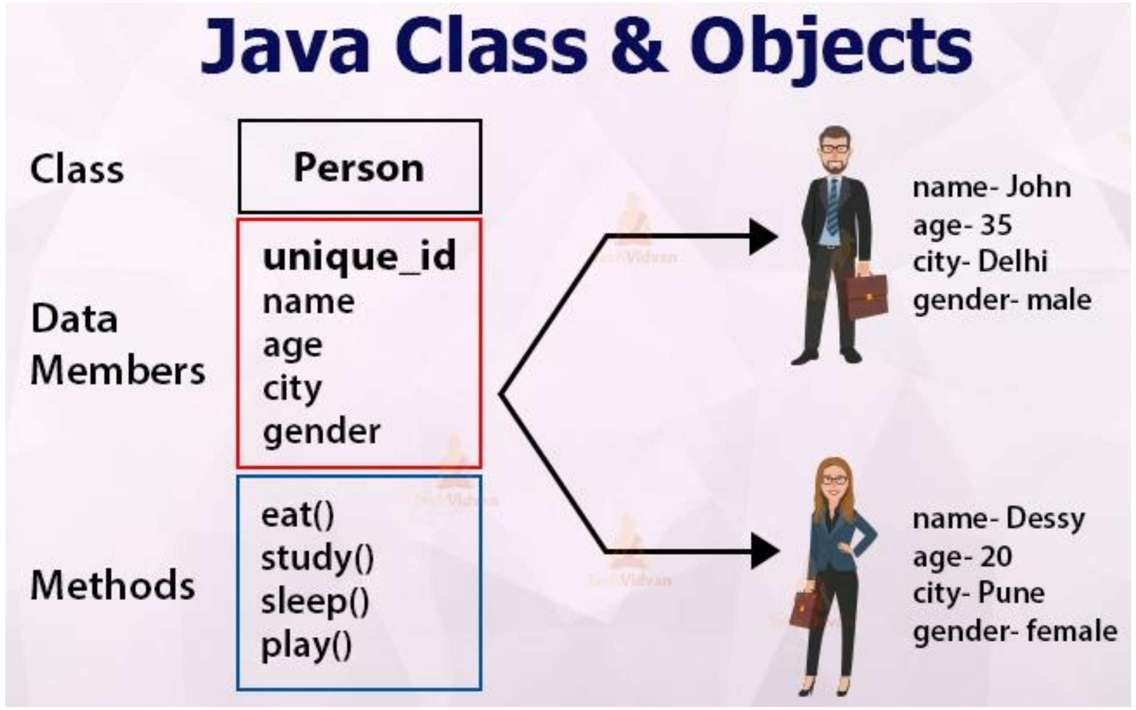
Class & Object



OOP

From Class to Object

Class & Object



OOP

From Class to Object

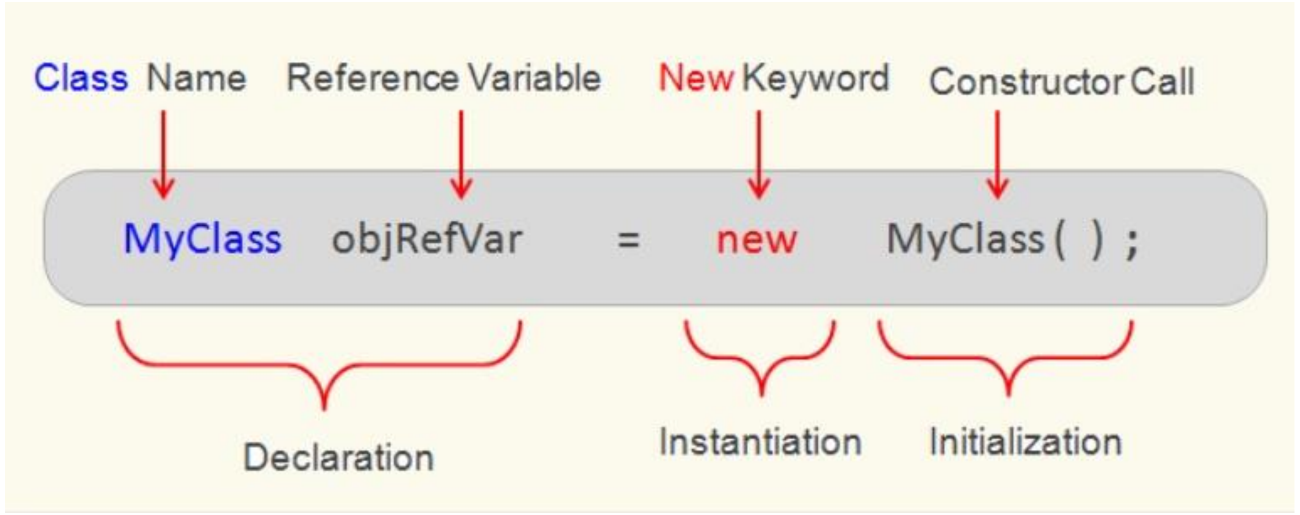
Constructor

```java
//default Constructor
public Person() {

}


//parameterize Constructor
public Person(int id, String firstName, String lastName) {

    this.id = id;
    this.firstName = firstName;
    this.lastName = lastName;
}



//parameterize Constructor
    public Person( String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
```

OOP

From Class to Object

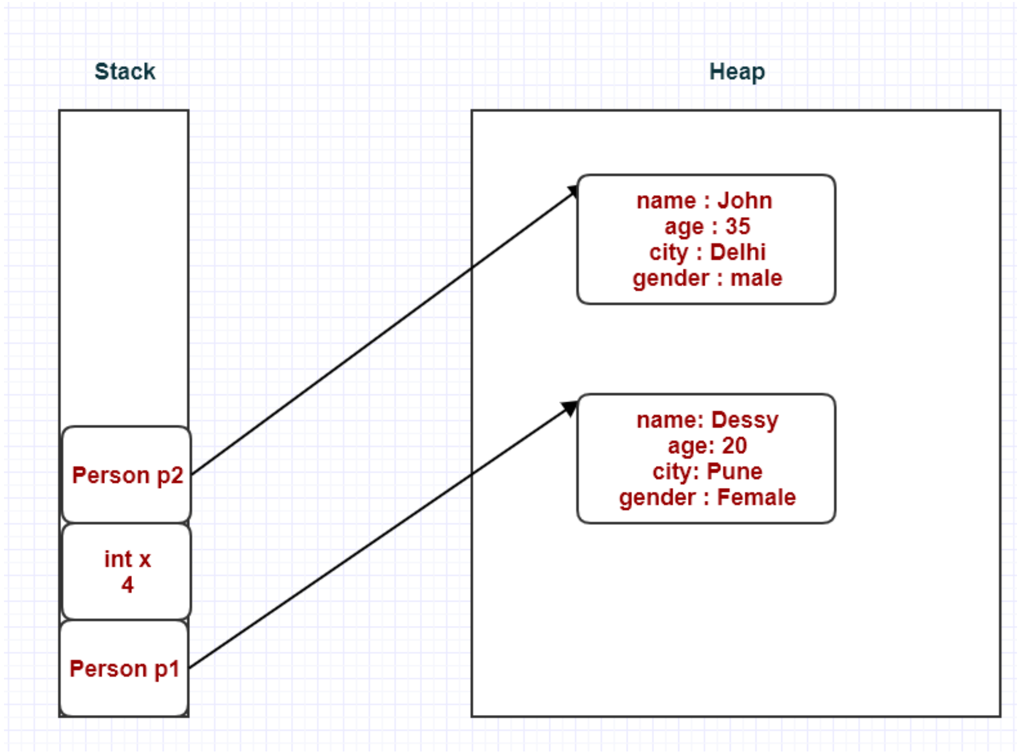
Class & Object



OOP

From Class to Object

Class & Object

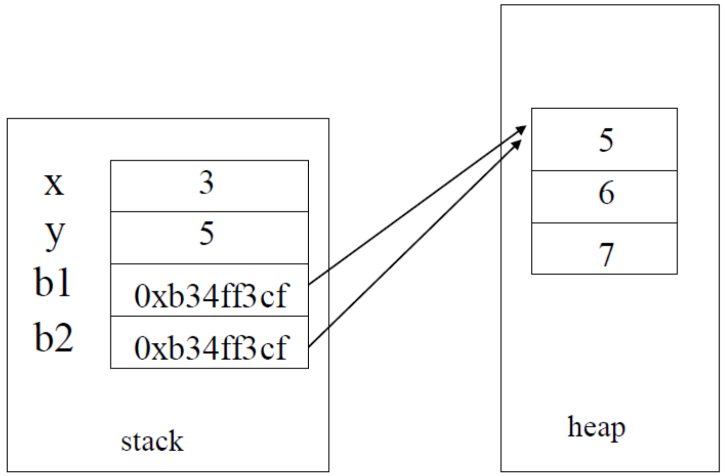


OOP

Initiates and values assignments

Class & Object

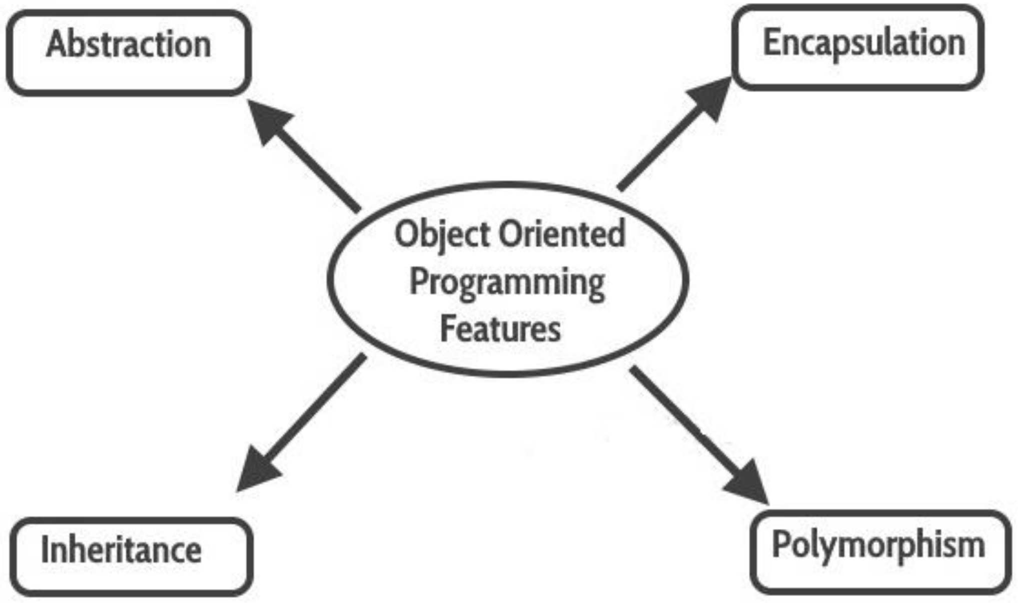
```
int x = 3, y = 5;
Box b1 = new Box(5, 6, 7);
Box b2 = b1;
```



OOP

OOP

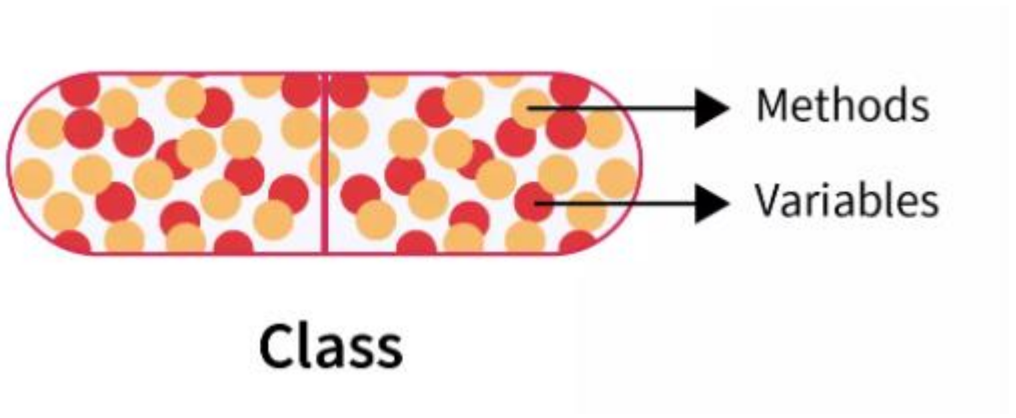
OOP Principles



OOP

Encapsulation

Encapsulation



OOP

What is an encapsulation ?

Encapsulation

- Encapsulation is the concept of bundling the data (variables) and the methods (functions) that operate on the data into a single unit, called an object.

- Internal object details are hidden from the outside world. Only specific, exposed methods can access and modify the object’s data.

- Public, private, and protected keywords are used to control the visibility of class members.

OOP

Encapsulation

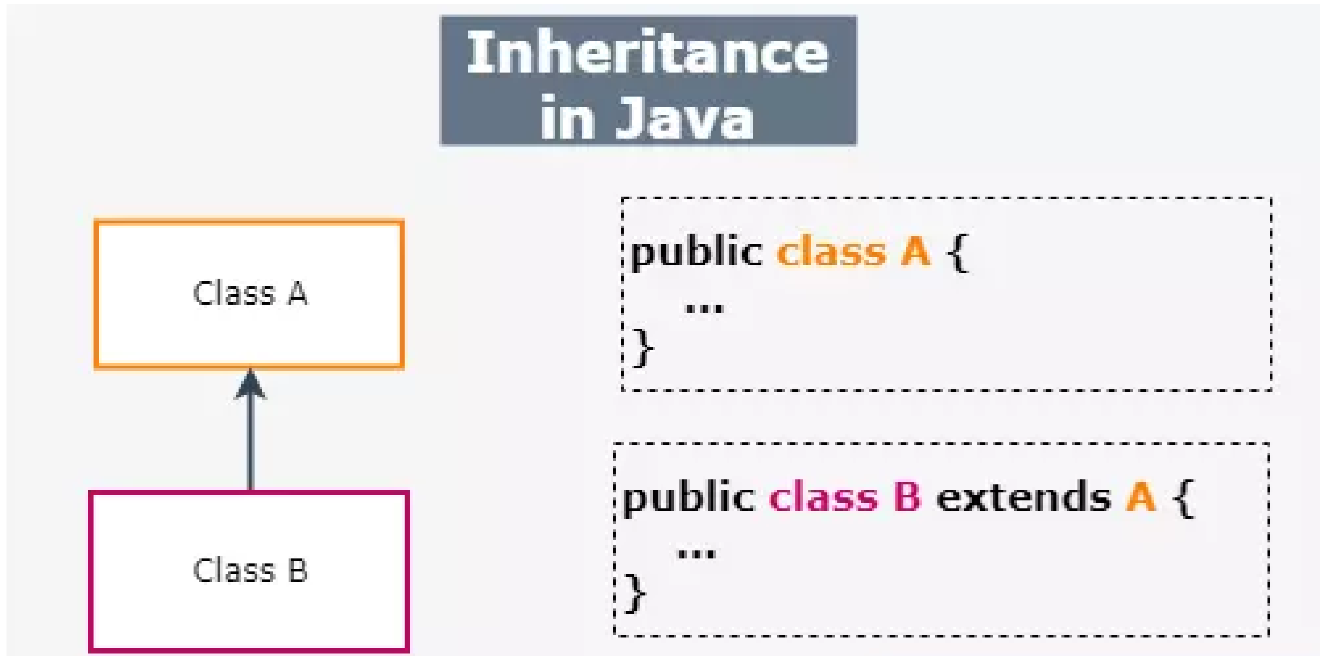
Example of encapsulation

```
public class Account {
private double balance; // Private variable

// Public method to get the balance
public double getBalance() {
return balance;
}

// Public method to deposit money
public void deposit(double amount) {
       if (amount > 0) {
              balance += amount;
       }
}

// Public method to withdraw money
public void withdraw(double amount) {
       if (amount > 0 &&     amount <= balance) {
              balance -= amount;
              }
       }
}
```

OOP

What is an Inheritance ?

Inheritance



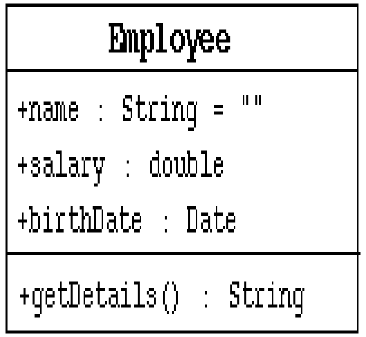
```
public class A {
    ...
}
```

```
public class B extends A {
    ...
}
```

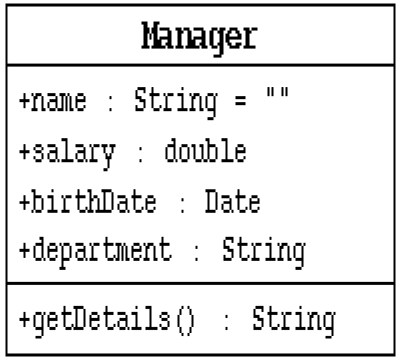
OOP

Example of Inheritance

Inheritance



```
public class Employee {
  public String name = "";
  public double salary;
  public Date birthDate;

  public String getDetails() {...}
}
```

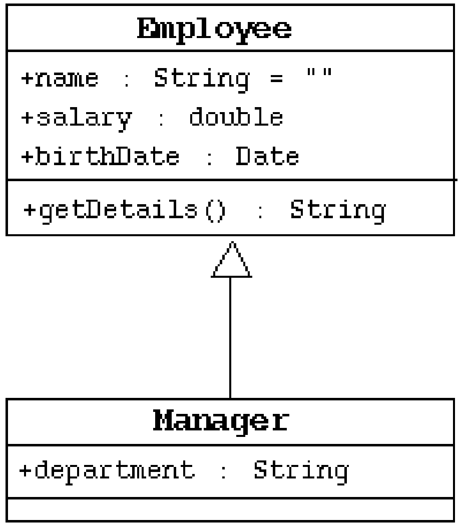


```
public class Manager {
  public String name = "";
  public double salary;
  public Date birthDate;
  public String department;

  public String getDetails() {...}
}
```

OOP

Example of Inheritance

Inheritance



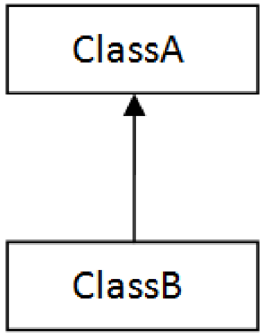
```
public class Employee {
  public String name = "";
  public double salary;
  public Date birthDate;

  public String getDetails() {...}
}
```

```
public class Manager extends Employee {
  public String department;
}
```

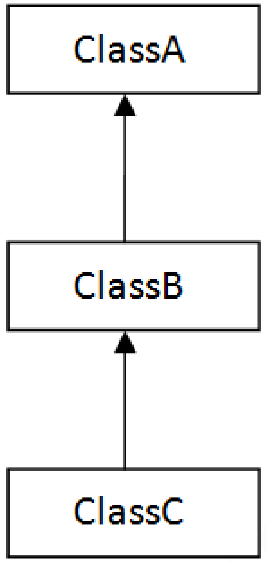
OOP

Allowed Inheritance

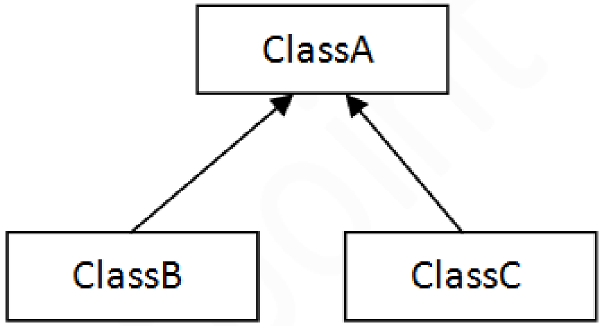
Inheritance



1) Single



2) Multilevel

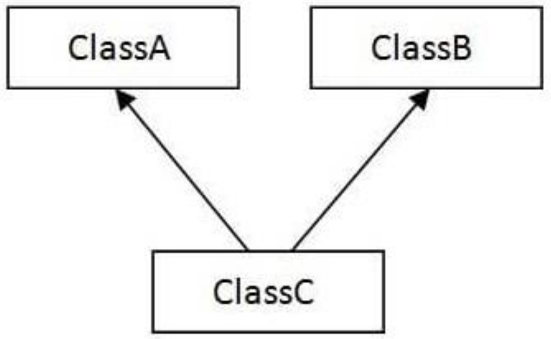


3) Hierarchical

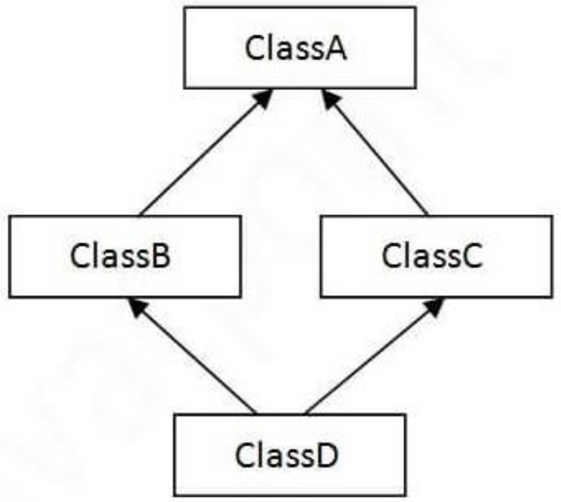
OOP

Not- Allowed Inheritance

Inheritance



4) Multiple



5) Hybrid

OOP

What is not inherited?

Inheritance

- Constructors
- Private attributes & operations (inherited but not reachable)
- Static attributes & operations

OOP

What is a Polymorphism ?

Polymorphism

- Polymorphism is the ability of an entity to several forms.
- In object-oriented programming, it refers t the ability of an object to take different fo of objects.

OOP

Polymorphism Types

Polymorphism

- Method polymorphism
- Object Polymorphism

OOP

Method Polymorphism

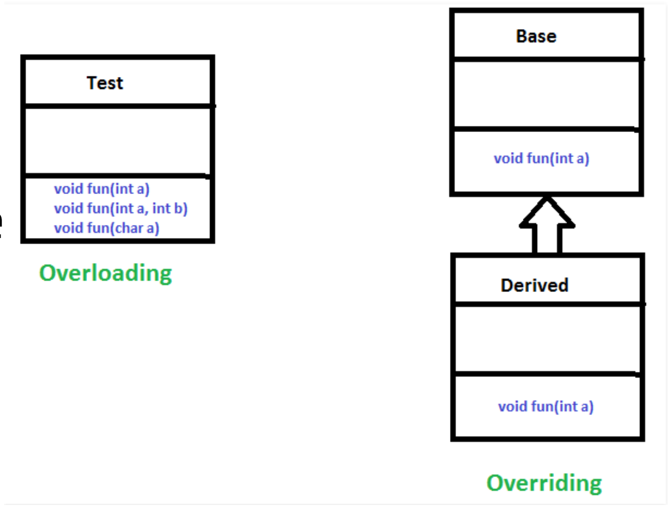
Polymorphism

- Overriding:

The Method fun is defined in the parent class base.
In the class Derived we are overriding the behavior of the method fun.

- Overloading:

In same class we are overloading fun (make some other copies)



OOP

Object Class

Polymorphism

Remember that the Object class is the root of all classes in Java

A class declaration with no extends clause, implicitly uses "extends Object"

Object's methods that are usually overridden are

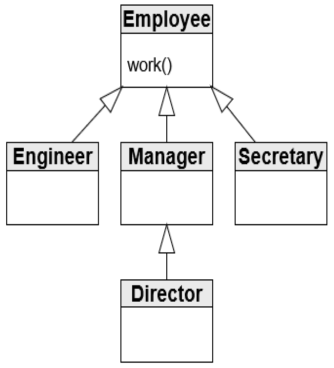
- toString
- equals
- hashCode

OOP

Objects Polymorphism

Polymorphism

- Subclass has all Functionality of Superclass.
- Subclass instance can be used in same place where super class instance is used.
- polymorphism we can make multi forms by creating deferent objects from deferent forms(classes) and refer them to the same reference variable



```
Employee [ ] emp = ...

emp [0] = new Employee();
emp [1] = new Engineer();
emp [2] = new Manager();
emp [3] = new Secretary();
emp [4] = new Director();

for (int i=0;i<emp.length;i=i+1){
    emp.work();
}
```

OOP

Objects

Polymorphism

Polymorphism

- Collections of objects with the same class type are called homogenous collections

- Collections of objects with different class types are called heterogeneous collections

```
Object[] objects = new
Object[5];
objects[0] = new Box(3, 2, 5);
objects[1] = "AAA";
objects[2] = new Date();
```

OOP

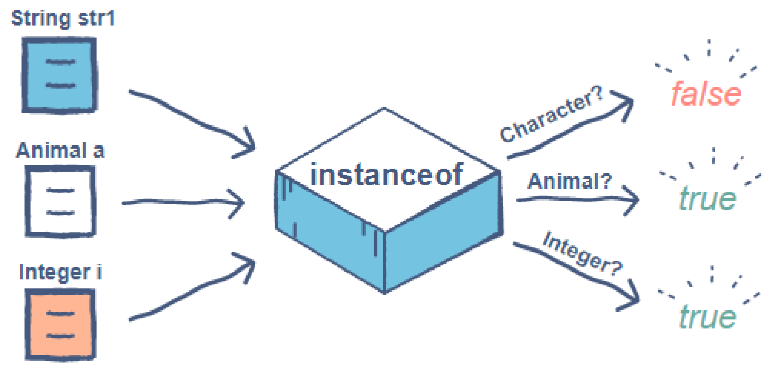
Polymorphism

Instance Of

instance of is an operator is used to check the type of
Return true when :

- instance referenced to the specific type of the Class

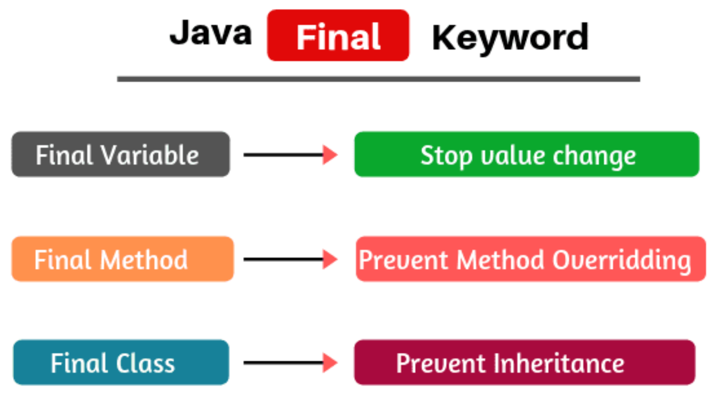
- instance referenced to the specific type of the superClass (inheritanc



OOP

Final & polymorphism

Polymorphism

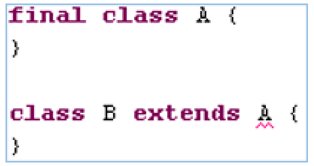


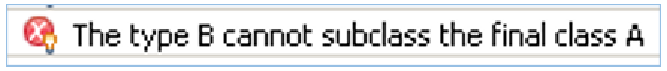
- Final class – cannot be inherited
  - Usually required for system classes
  - Some basic behaviors that mustn't be extended or changed
- Final method cannot be overridden
  - Forces sub-classes to use a specific implementation
- Final variable can have only one assignment [constant]
  - Defines a constant values
  - Local variables can also be defined as final

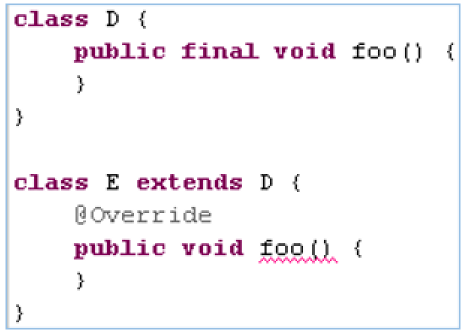
OOP

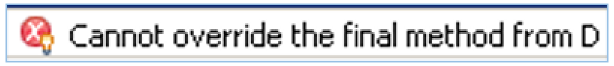
Final & polymorphism

Polymorphism


```
final class A {
}

class B extends A {
}
```


The type B cannot subclass the final class A


```
class D {
    public final void foo() {
    }
}

class E extends D {
    @Override
    public void foo() {
    }
}
```


Cannot override the final method from D

OOP

Abstract class

Abstraction

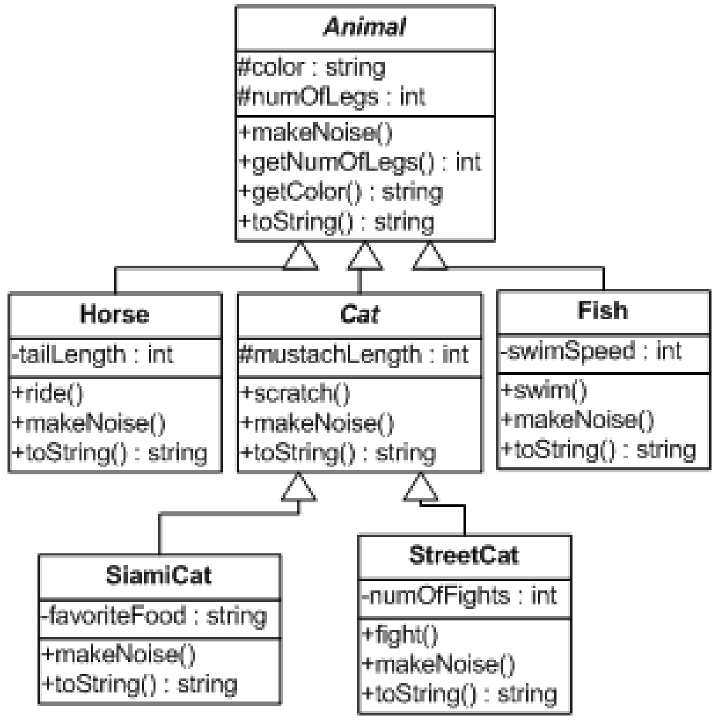
- A class that represents an abstract object and therefore can not be instantiated .
- Abstract classes may(but not mandatory) have abstract methods.

- A method that represents an abstract operation and therefore has no body (implementation).
- These method must be overridden in non-abstract subclasses to provide an implementation.
- A class /method cannot be both final and abstract at the same time

OOP

Abstraction

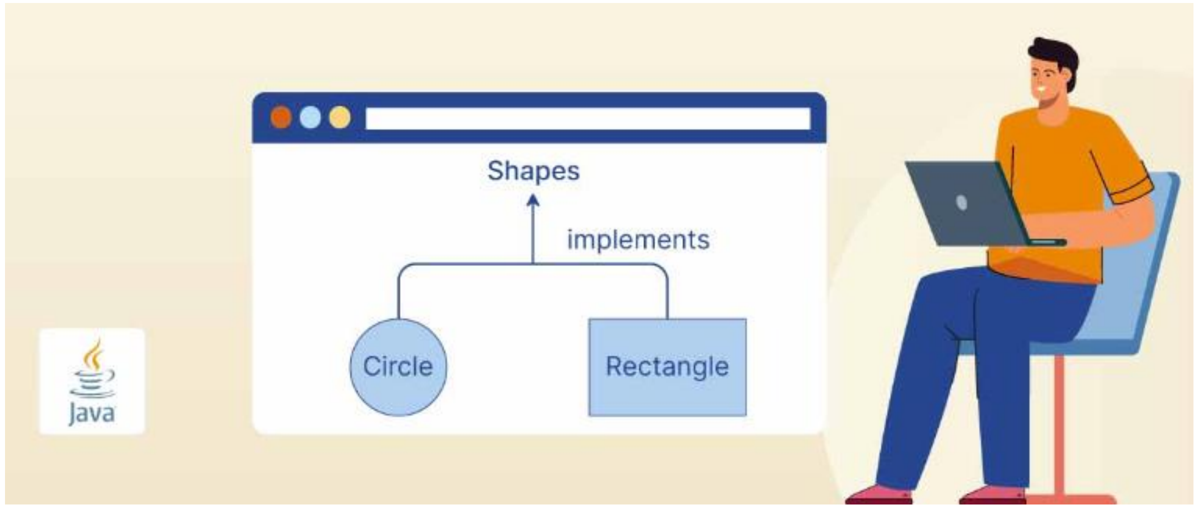
Abstract Example



OOP

What are interfaces ?

Interfaces



OOP

What are interfaces ?

Interfaces

- A "public interface" is a contract between client code and the class that implements that interface.
- A Java interface is a formal declaration of such contract in which all methods contain no implementation.
- Many, unrelated classes can implement the same interface .
- A class can implement many, unrelated interfaces

OOP

What are interfaces ?

Interfaces

Interfaces can contain:

- final variables (constants)
- abstract methods
- static methods
- default methods (will be discussed later)

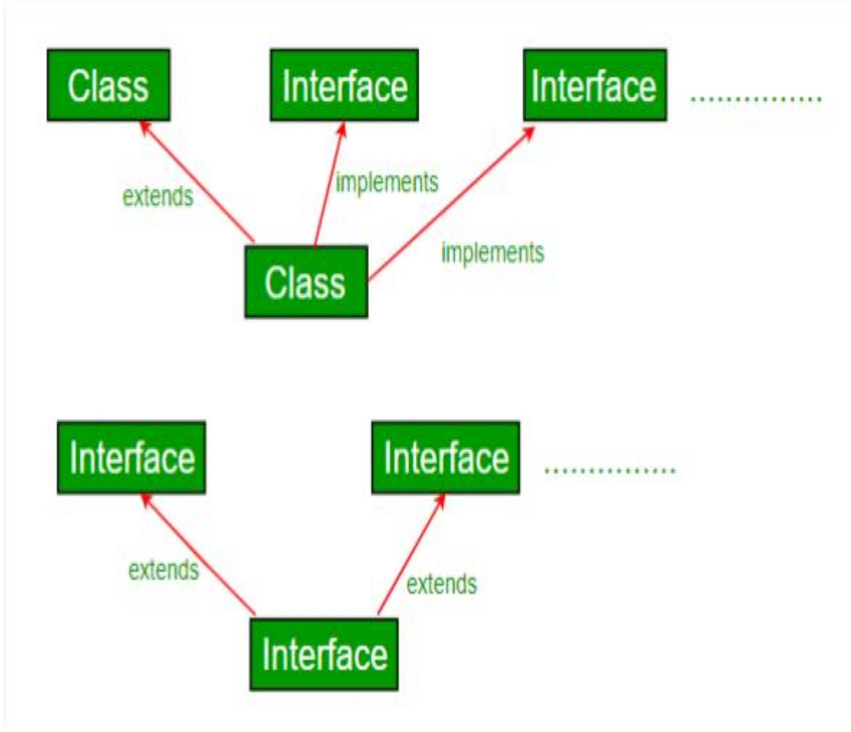
A non-abstract class implements an interface if and only if:

- it declares that it implements the interface (using the implements keyword)
- it implements all of the abstract methods of the interface

OOP

What are interfaces ?

Interfaces

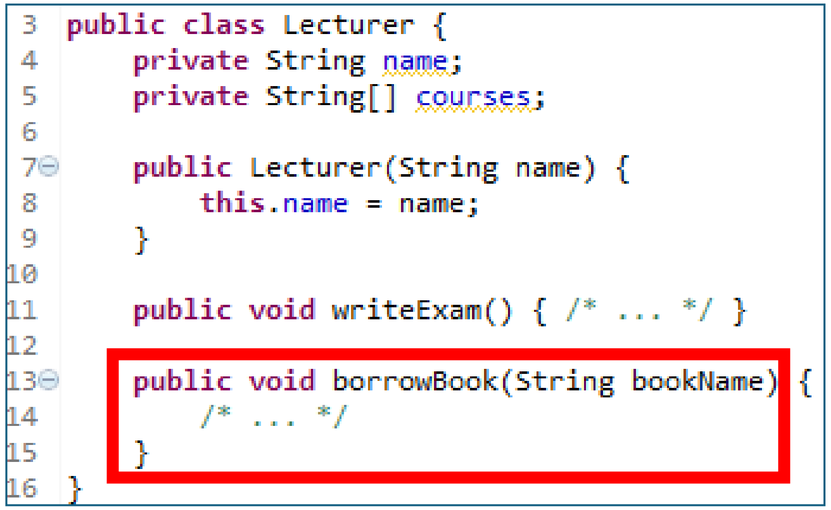


OOP

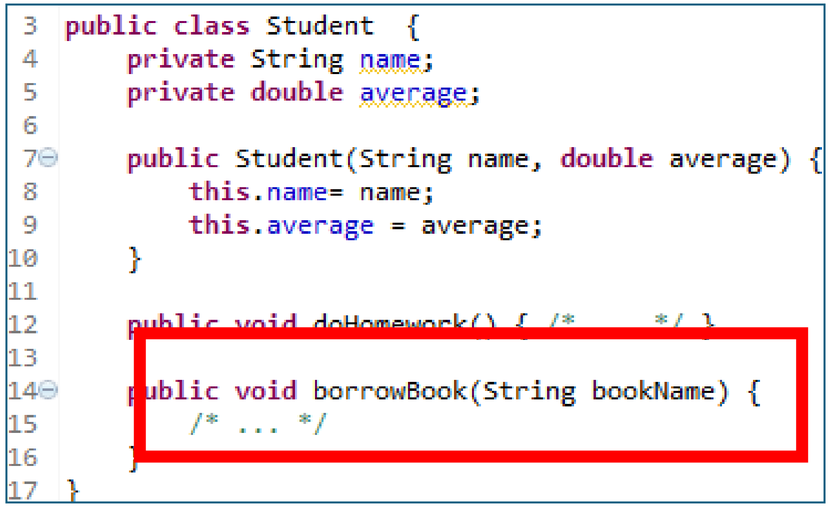
What are interfaces ?

Interfaces

לשתי המחלקות יש במקרה פעולה
זהה, אבל מסיבה כלשהי איננו רוצים
שיהיה אבא משותף



```java
public class Lecturer {
    private String name;
    private String[] courses;

    public Lecturer(String name) {
        this.name = name;
    }

    public void writeExam() { /* ... */ }

    public void borrowBook(String bookName) {
        /* ... */
    }
}
```

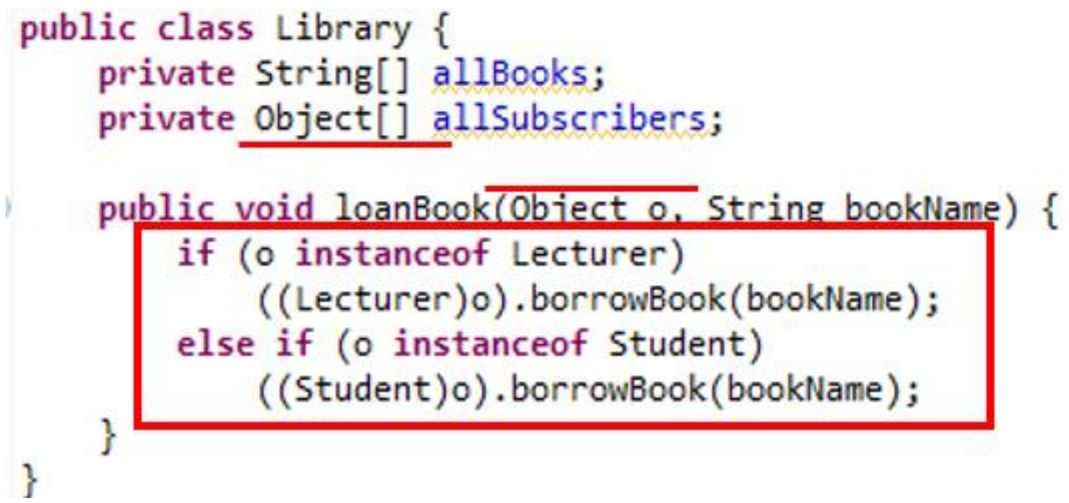


```java
public class Student  {
    private String name;
    private double average;

    public Student(String name, double average) {
        this.name= name;
        this.average = average;
    }

    public void doHomework() { /* ... */ }

    public void borrowBook(String bookName) {
        /* ... */
    }
}
```

OOP

What are interfaces ?

Interfaces



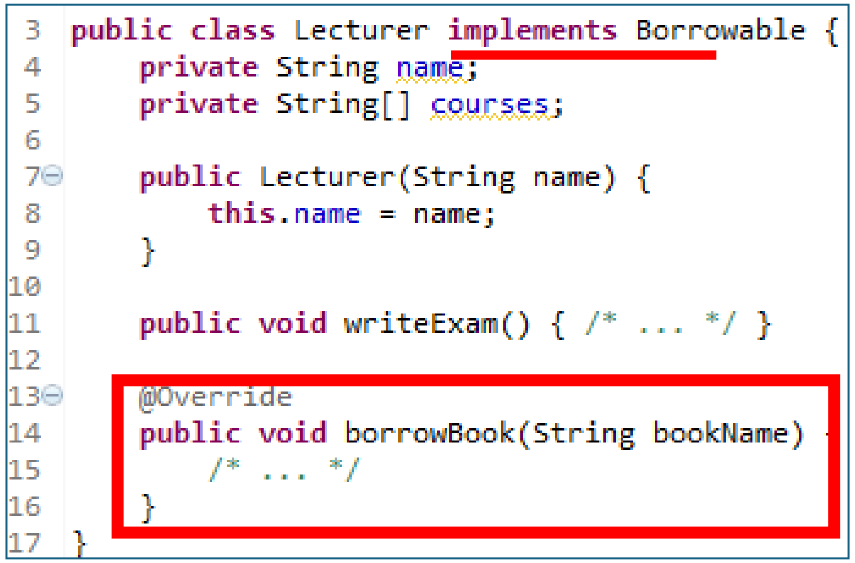
```
public class Library {
    private String[] allBooks;
    private Object[] allSubscribers;

    public void loanBook(Object o, String bookName) {
        if (o instanceof Lecturer)
            ((Lecturer)o).borrowBook(bookName);
        else if (o instanceof Student)
            ((Student)o).borrowBook(bookName);
    }
}
```

מאחר ואובייקטים מסוגים שונים, ללא אבא משותף, יכולים להיות פרמטרים למתודה או חלק ממערך המחלקה, היא חייבת לקבל Object

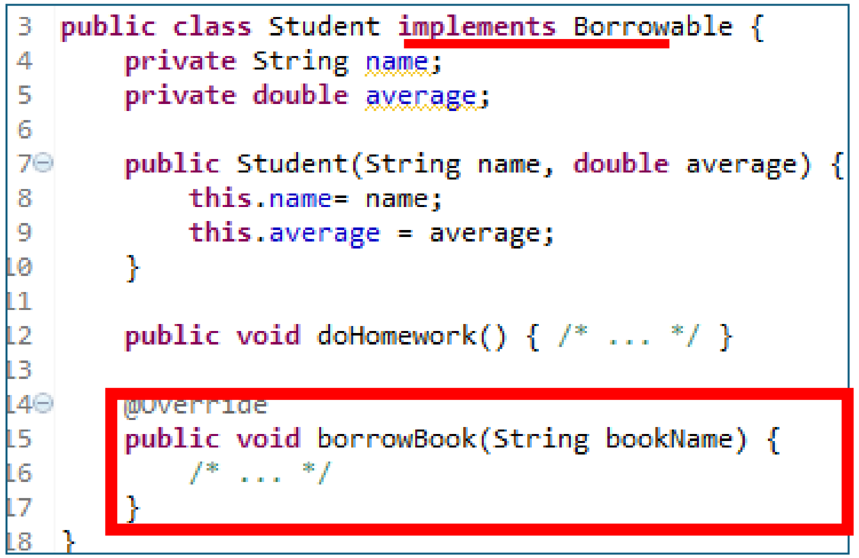
OOP

What are interfaces ?

Interfaces



```java
public class Lecturer implements Borrowable {
    private String name;
    private String[] courses;

    public Lecturer(String name) {
        this.name = name;
    }

    public void writeExam() { /* ... */ }

    @Override
    public void borrowBook(String bookName)
        /* ... */
    }
}
```

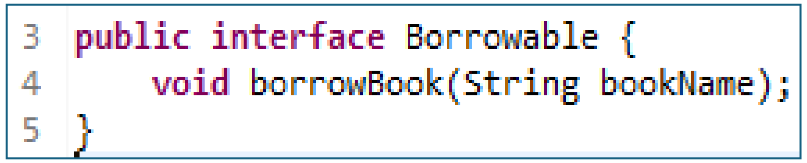


```java
public class Student implements Borrowable {
    private String name;
    private double average;

    public Student(String name, double average) {
        this.name= name;
        this.average = average;
    }

    public void doHomework() { /* ... */ }

    @Override
    public void borrowBook(String bookName) {
        /* ... */
    }
}
```

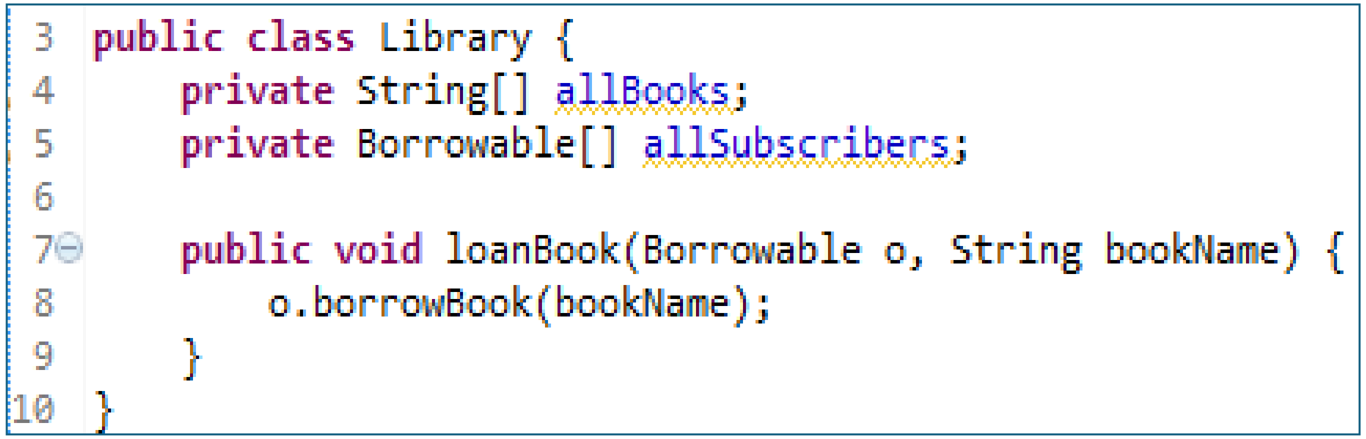
OOP

What are interfaces ?

Interfaces



```java
public interface Borrowable {
    void borrowBook(String bookName);
}
```

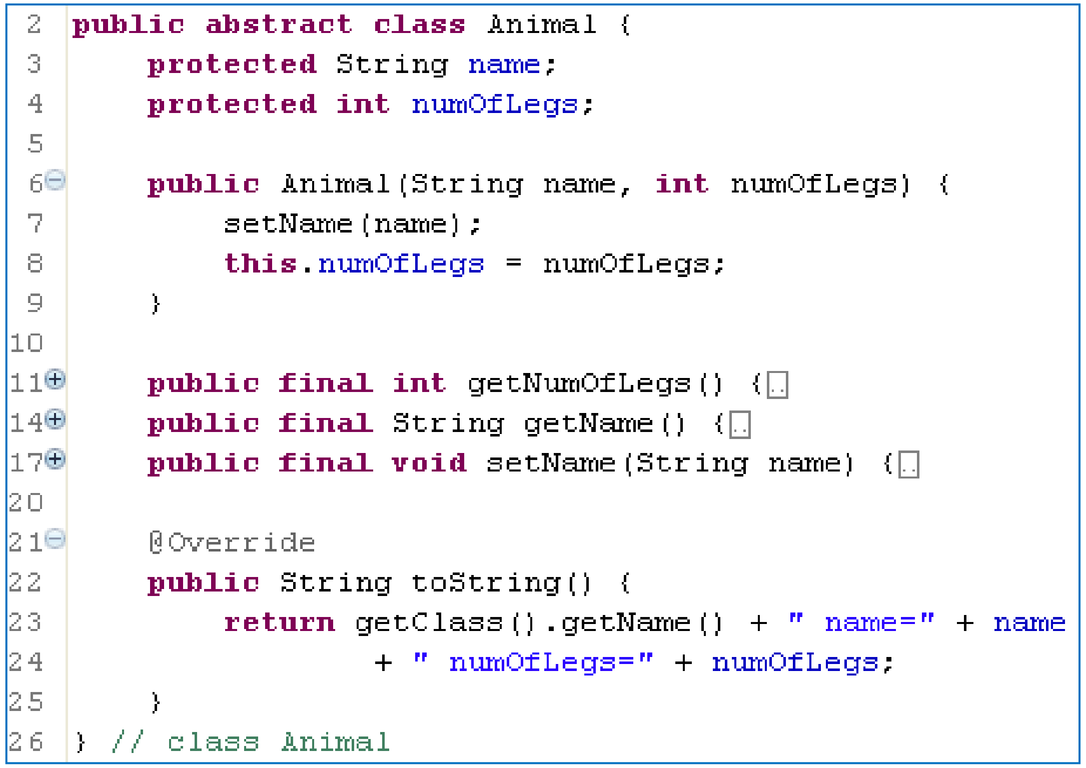


```java
public class Library {
    private String[] allBooks;
    private Borrowable[] allSubscribers;

    public void loanBook(Borrowable o, String bookName) {
        o.borrowBook(bookName);
    }
}
```

OOP

What are interfaces ?

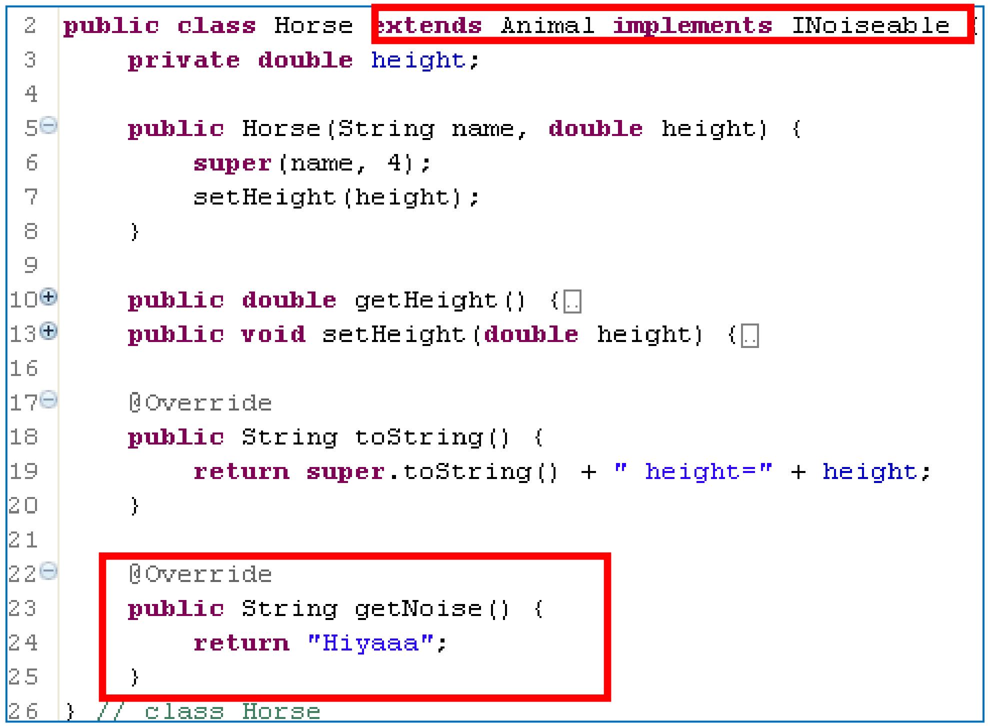
Interfaces


```java
public abstract class Animal {
    protected String name;
    protected int numOfLegs;

    public Animal(String name, int numOfLegs) {
        setName(name);
        this.numOfLegs = numOfLegs;
    }

    public final int getNumOfLegs() {...
    public final String getName() {...
    public final void setName(String name) {...

    @Override
    public String toString() {
        return getClass().getName() + " name=" + name
                + " numOfLegs=" + numOfLegs;
    }
} // class Animal
```

OOP

What are interfaces ?

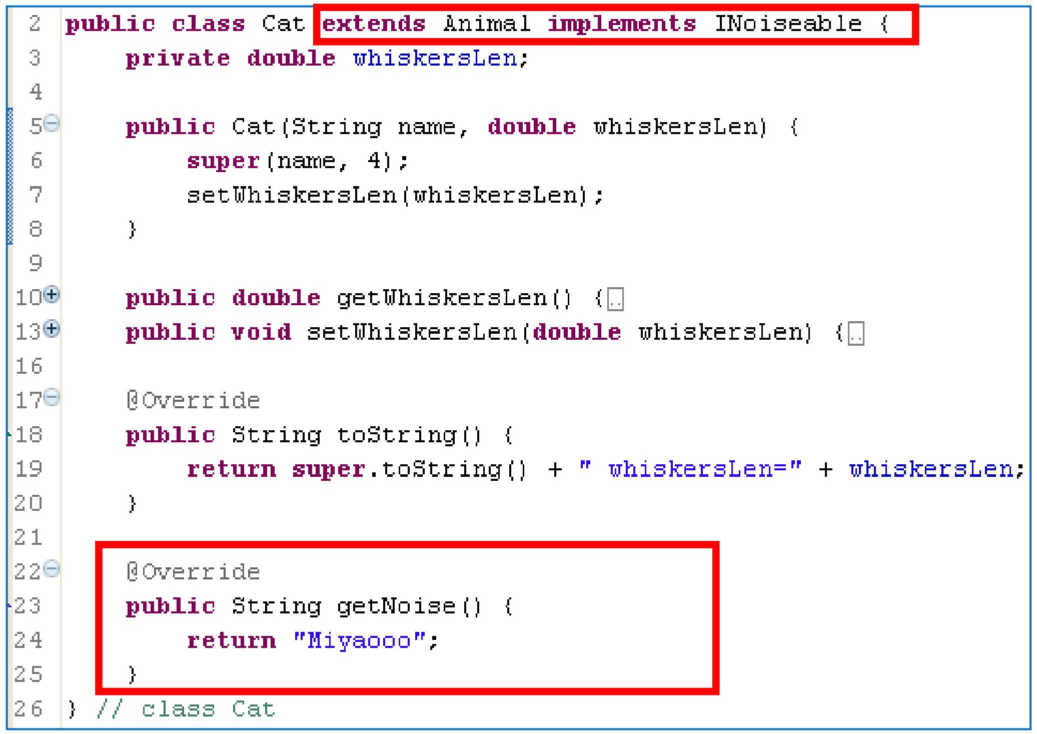
Interfaces


```
public class Horse extends Animal implements INoiseable
    private double height;

    public Horse(String name, double height) {
        super(name, 4);
        setHeight(height);
    }

    public double getHeight() {
    public void setHeight(double height) {

    @Override
    public String toString() {
        return super.toString() + " height=" + height;
    }

    @Override
    public String getNoise() {
        return "Hiyaaa";
    }
} // class Horse
```

OOP

What are interfaces ?

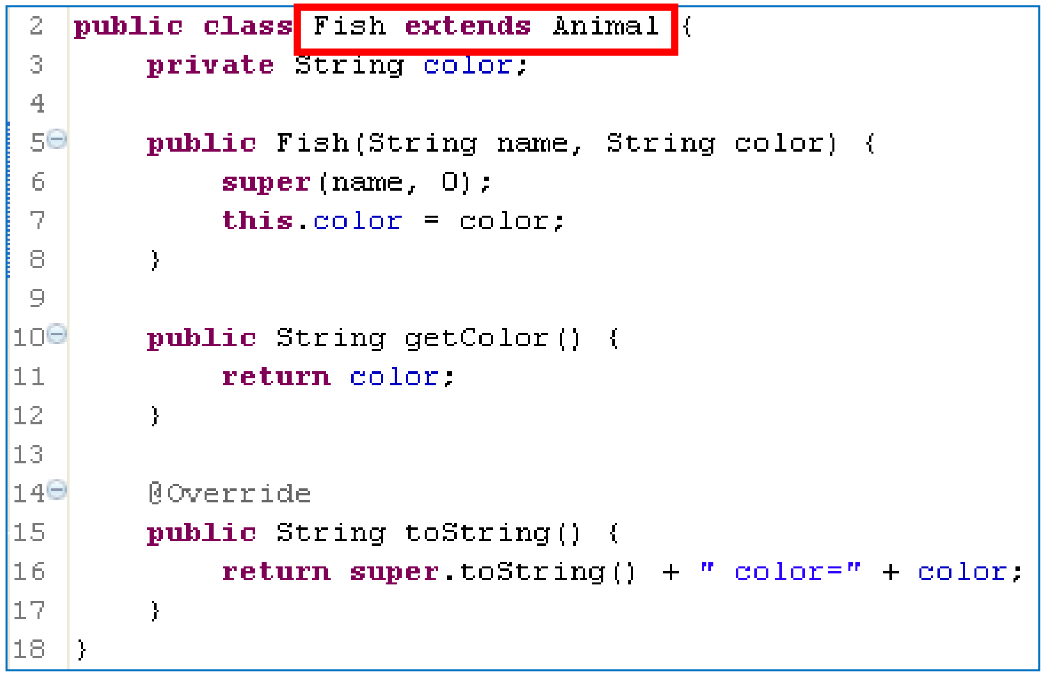
Interfaces


```
public class Cat extends Animal implements INoiseable {
    private double whiskersLen;

    public Cat(String name, double whiskersLen) {
        super(name, 4);
        setWhiskersLen(whiskersLen);
    }

    public double getWhiskersLen() {
    public void setWhiskersLen(double whiskersLen) {

    @Override
    public String toString() {
        return super.toString() + " whiskersLen=" + whiskersLen;
    }

    @Override
    public String getNoise() {
        return "Miyaooo";
    }
} // class Cat
```

OOP

What are interfaces ?

Interfaces

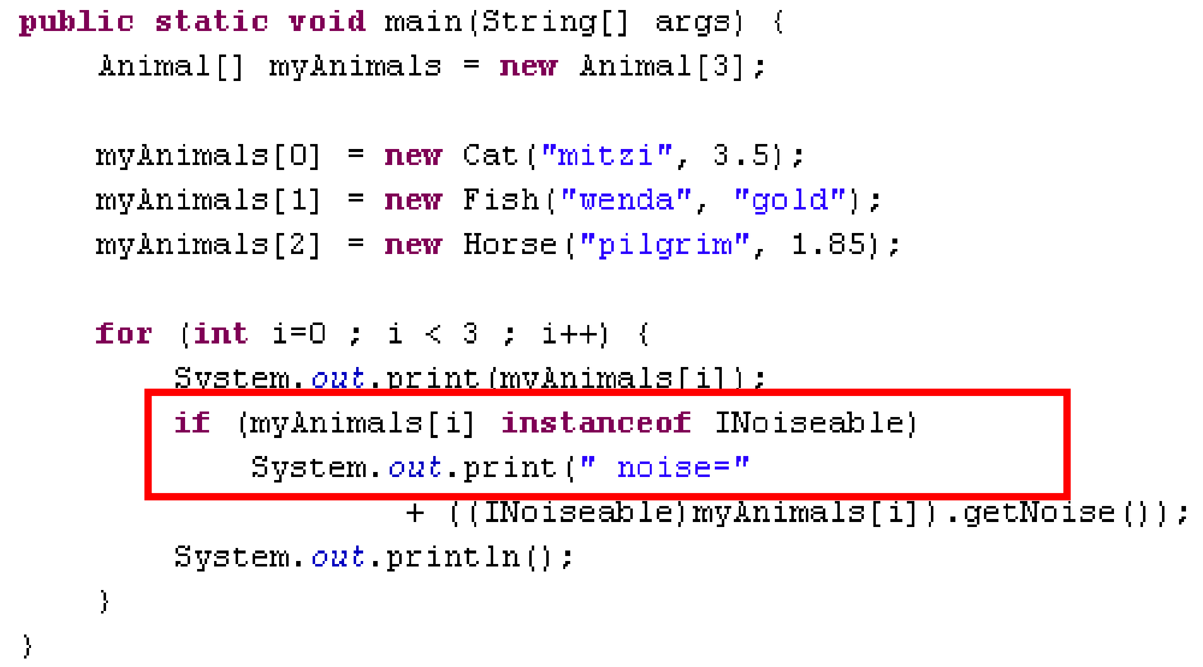


```
public class Fish extends Animal {
    private String color;

    public Fish(String name, String color) {
        super(name, 0);
        this.color = color;
    }

    public String getColor() {
        return color;
    }

    @Override
    public String toString() {
        return super.toString() + " color=" + color;
    }
}
```

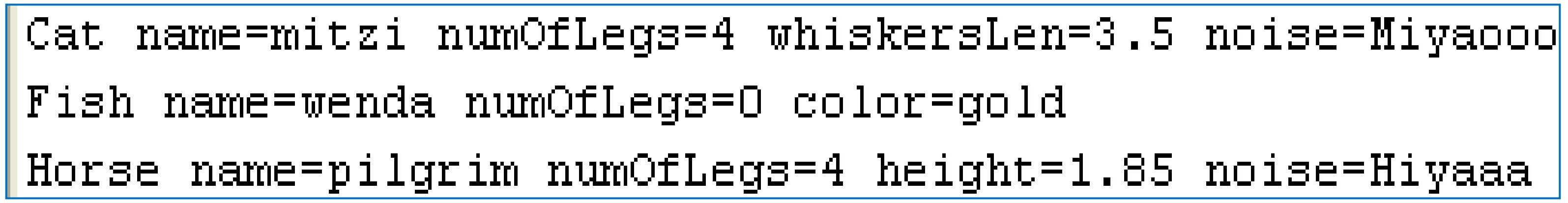
OOP

Interfaces

What are interfaces ?



```
public static void main(String[] args) {
    Animal[] myAnimals = new Animal[3];

    myAnimals[0] = new Cat("mitzi", 3.5);
    myAnimals[1] = new Fish("wenda", "gold");
    myAnimals[2] = new Horse("pilgrim", 1.85);

    for (int i=0 ; i < 3 ; i++) {
        System.out.print(myAnimals[i]);
        if (myAnimals[i] instanceof INoiseable)
            System.out.print(" noise="
                    + ((INoiseable)myAnimals[i]).getNoise());
        System.out.println();
    }
}
```



```
Cat name=mitzi numOfLegs=4 whiskersLen=3.5 noise=Miyaooo
Fish name=wenda numOfLegs=0 color=gold
Horse name=pilgrim numOfLegs=4 height=1.85 noise=Hiyaaa
```

OOP

Interfaces

Comparable

- An interface used to define the natural ordering of objects.
- Includes only method signature compareTo(Object o).
- Implemented by a class whose objects need to be ordered.

```
public class Student implements Comparable<Student> {
private int id;
private String name;


public int compareTo(Student other) {
return this.id - other.id; // Natural ordering by ID
 } }
```

OOP

Interfaces

Comparator

- An interface used to define the custom ordering of objects.
- Includes only method signature compare(Object o1, Object o2)
- Implemented by a separate class or through lambda expressions to provide multiple ways to compare objects.

```
public class Student implements Comparator<Student> {

public int compare(Student s1, Student s2) {

return s1.getName().compareTo(s2.getName());
// Custom ordering by name
        }
}
```

OOP

OOP

Thank You !!