Java Programming
Unit 4

Casting, Abstract Classes, Interfaces, Polymorphism

Interfaces

- *Interfaces* can contain only declarations of methods and final variables.

```
public interface Payable {
        boolean increasePay(int percent);
    }
```

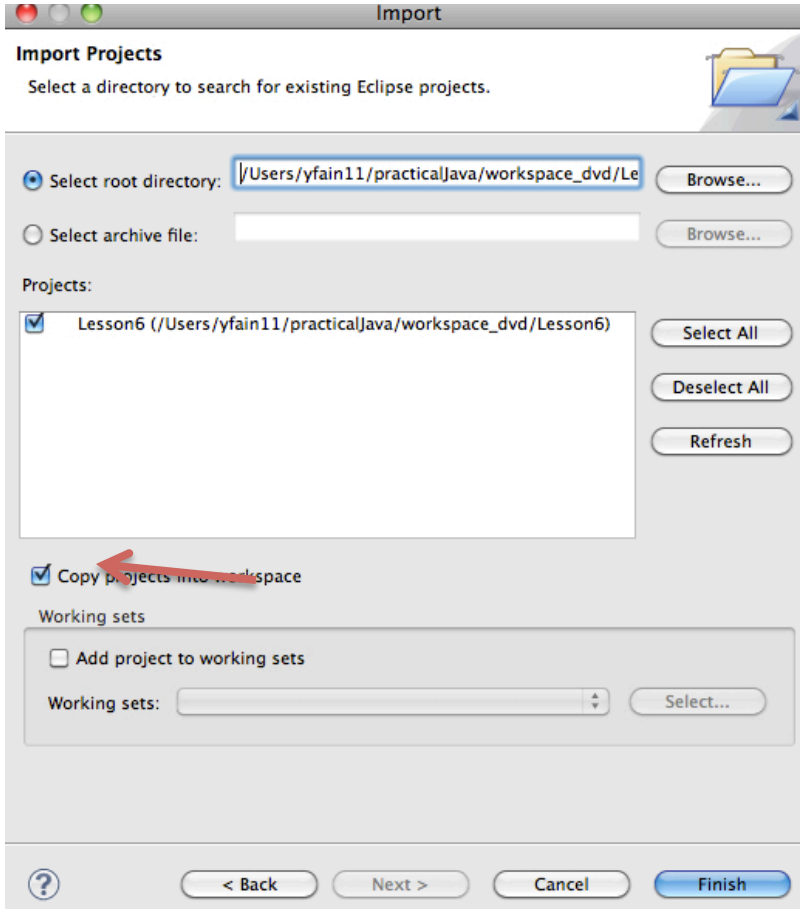
- A class can implement one or more interfaces

```
class Employee implements Payable, Promotionable {…}

class Contractor implements Payable{…}
```

- If a class declaration has the `implements` keyword it MUST implement every method that's declared in the interface(s) that this class implements.

Walkthrough 1

1. Download the project Lesson6 from the Textbook Website http://bit.ly/cEegvC .
2. Import it to Eclipse: File | Import |General | Existing projects into workspace). Check off the option Copy projects into workspace.



3. Examine with the instructor the structure of the project.

4. Review the code of classes Person, Employee, and TestPayIncrease.

5. Run the program TestPayIncrease located in the default package.

Casting

All Java classes form an inheritance tree with the class `Object`. While declaring non-primitive variables you are allowed to use either the exact data type of this variable or one of its ancestor data types. For example, if the class `NJTax` extends `Tax` each of these lines is correct.

```
NJTax myTax1    = new NJTax();
Tax myTax2      = new NJTax();    // upcasting
Object myTax3   = new NJTax();    // upcasting
```

If `Employee` and `Contractor` extend class `Person`, you can declare array of type `Person`, but populate it with employees and contractors:

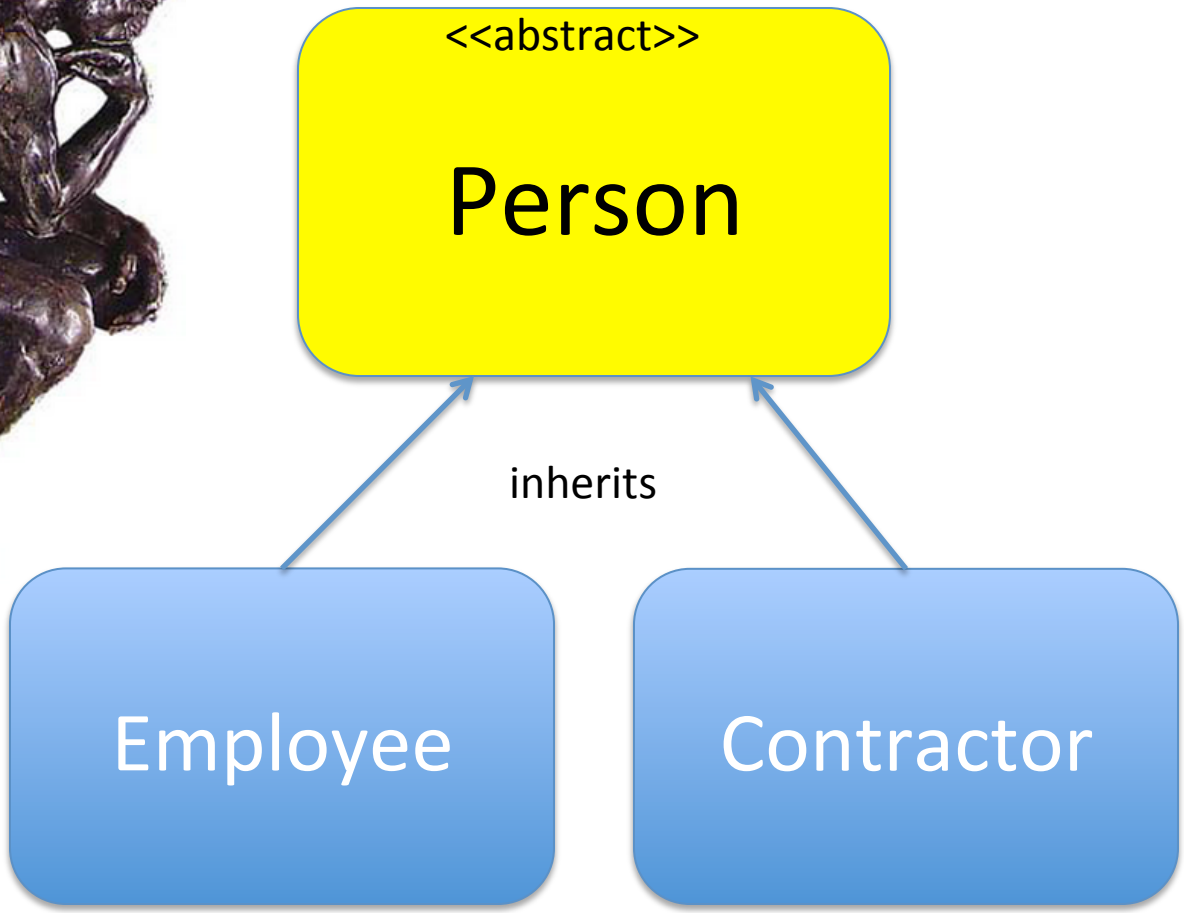
```
Person workers[] = new Person [100];

workers[0] = new Employee("Yakov", "Fain");
workers[1] = new Employee("Mary", "Lou");
workers[2] = new Contractor("Bill", "Shaw");
```

Casting (cont.)

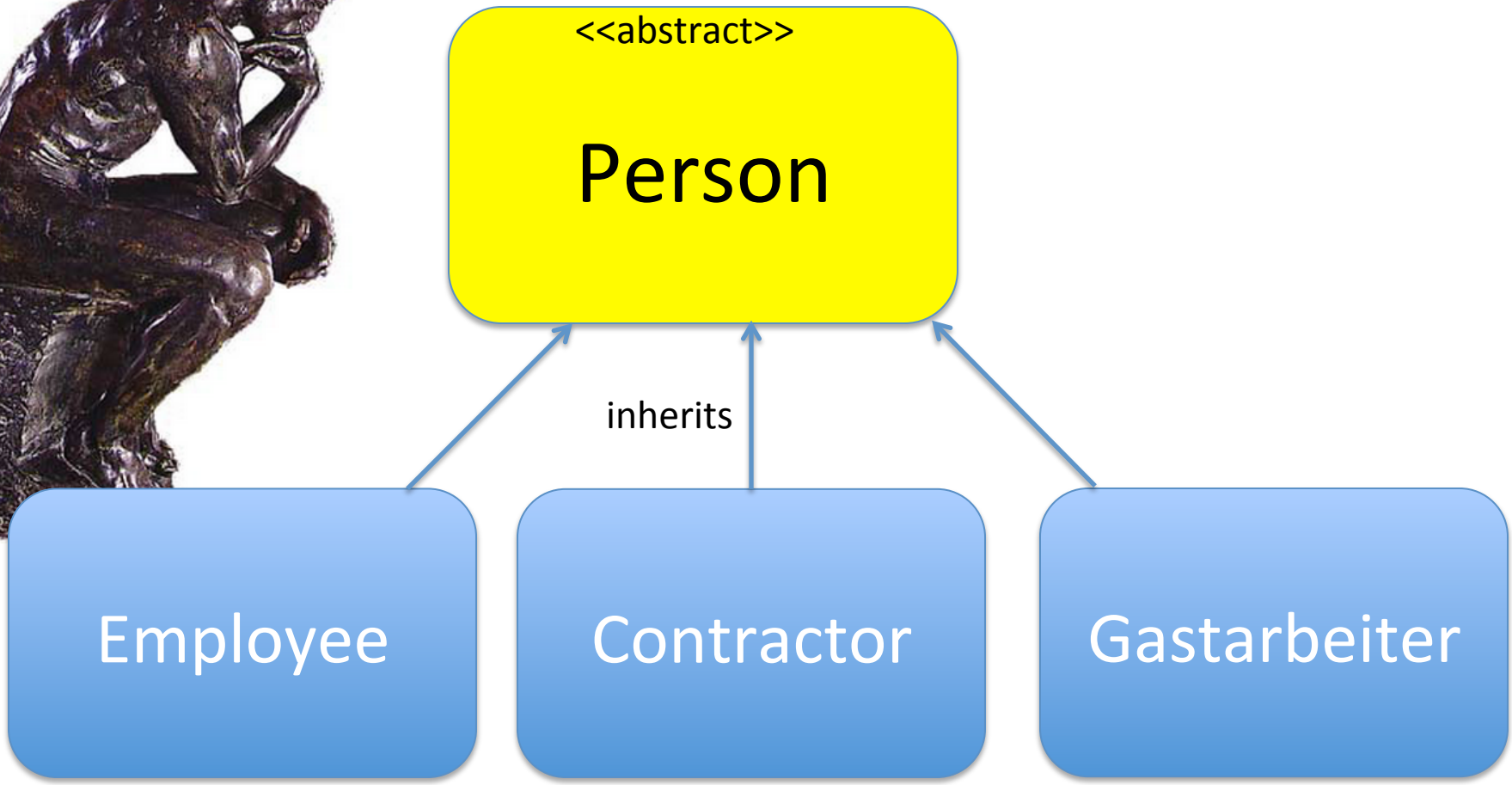
While processing a collection of different objects you may use the `instanceof` operator to check the actual data type of an object. Placing a data type in parenthesis in front of another type means that you want to *cast* this object to specified type.

```
Person workers[] = new Person [20];

// Populate the array workers here….
for (int i=0; i<20; i++){
     Employee currentEmployee;
     Contractor currentContractor;

     if (workers[i] instanceof Employee){            // type check

          currentEmployee = (Employee) workers[i];   // downcasting
          // do some employee-specific processing here

     } else if  (workers[i] instanceof Contractor){

         currentContractor = (Contractor) workers[i];    // downcasting
         // do some contractor-specific  processing here
     }
}
```

Java Team Lead at Work



Java Team Lead at Work



Abstract Classes

A class is called abstract if it was declared with the `abstract` keyword.

You can not instantiate an abstract class. Usually, an abstract class has at least one abstract method.

```
abstract public class Person {

    public void changeAddress(String address){
       System.out.println("New address is" + address);
    }
 ...
  // an abstract method to be implemented in subclasses
   public abstract boolean increasePay(int percent);
}
```

The increasePay() method must be implemented in one of the subclasses of Person.

Abstract == Unfinished

An abstract class is a sketch of the future concrete class.

If a subclass of an abstract class doesn't implement all abstract methods it remains abstract.

Promoting Workers. Take 1.

A company has employees and contractors. Design the classes without ***using interfaces*** to represent people who work for this company.

The classes should have the following methods:

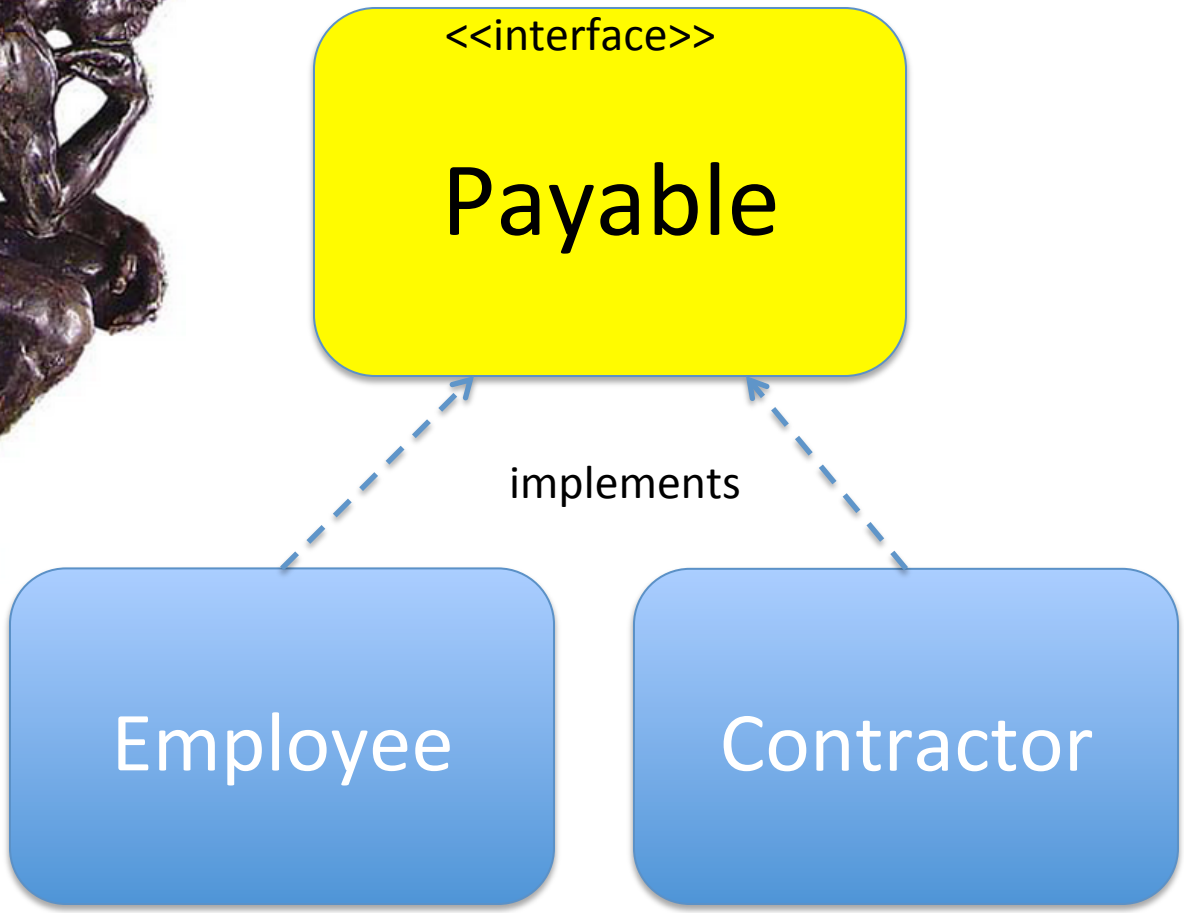
```
changeAddress()
promote()
giveDayOff()
increasePay()
```

Promotion means giving one day off and raising the amount in the pay check.

For employees, the method increasePay() should raise the yearly salary.

For contractors, the method increasePay() should increase their hourly rate.

Java Team Lead at Work



```java
abstract public class Person {

        private String name;
        int INCREASE_CAP = 20;  // cap on pay increase

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

    public String getName(){
        return "Person's name is " + name;
    }

    public void changeAddress(String address){
            System.out.println("New address is" + address);
    }

    private void giveDayOff(){
            System.out.println("Giving a day off to " + name);
    }

      public void promote(int percent){
            System.out.println(" Promoting a worker...");
            giveDayOff();

            //call the abstract method increasePay(percent) here
     }
     // an abstract method to be implemented in subclasses
     public abstract boolean increasePay(int percent);
}
```

Listing 7.1 from the text book shows an abstract ancestor

```java
public class Employee extends Person{

    public Employee(String name){
        super(name);
    }
    public boolean increasePay(int percent) {
        System.out.println("Increasing salary by " +
                    percent + "%. "+ getName());

            return true;
    }
}
```

Descendants of Person implement increasePay() differently

```java
public class Contractor extends Person {

    public Contractor(String name){
        super(name);
    }
    public boolean increasePay(int percent) {
        if(percent < INCREASE_CAP){
            System.out.println("Increasing hourly rate by " +
                                percent + "%. "+ getName());
            return true;
        } else {
            System.out.println("Sorry, can't increase hourly rate by
                    more than ” + INCREASE_CAP + "%. "+ getName());
            return false;
        }
    }
}
```

An Example of Polymorphism. Increasing Pay to Every Worker.

```
public class TestPayIncrease2 {

    public static void main(String[] args) {

        Person workers[] = new Person[3];

         workers[0] = new Employee("John");
         workers[1] = new Contractor("Mary");
         workers[2] = new Employee("Steve");

         for (Person p: workers){
                  p.promote(30);
         }
    }
}
```

The array workers has a mix of employees and contractors, but the class TestPayIncrease2 promotes workers without checking if the current instance is Employee or Contractor.

The proper version of the method promote() will be invoked based on the actual type of the worker. This is an illustration of a polymorphic behavior.

Walkthrough 2

1. Download the project Lesson7 from the Textbook Website http://bit.ly/YmZtph.

2. Import the project Lesson7 to Eclipse.

3. Run the program TestPayIncrease2.
4. The output of TestPayIncrease is shown below. Try to understand it. Ask questions.

```
Promoting a worker...
Giving a day off to John
Increasing salary by 30%. Person's name is John
 Promoting a worker...
Giving a day off to Mary
Sorry, can't increase hourly rate by more than 20%. Person's name is Mary
 Promoting a worker...
Giving a day off to Steve
Increasing salary by 30%. Person's name is Steve
```

A polymorphic solution with interfaces

```
public class TestPayIncreasePoly {

   public static void main(String[] args) {

        Payable workers[] = new Payable[3];

         workers[0] = new Employee("John");
         workers[1] = new Contractor("Mary");
         workers[2] = new Employee("Steve");

         for (Payable p: workers){
            p.increasePay(30);
         }
   }
}
```

Assumption:
both Employee and Contractor implement Payable (see Listing 6-2 and 6-3) in text book.

The variable p can be used to invoke only the methods defined in the interface Payable regardless of how many methods declared in classes Employee and Contractor.

Homework

1. Study the materials from Lesson 7 from the textbook and do the assignment from its Try It section.

2. Invent and program any sample application that can be implemented with interfaces illustrating polymorphism. For example, think of the classes Cat and Dog, Man and Woman, or a store inventory that has to be discounted…

Additional Reading

Java Interfaces:
http://docs.oracle.com/javase/tutorial/java/concepts/interface.html

Abstract Classes:
http://docs.oracle.com/javase/tutorial/java/IandI/abstract.html

UML Refcard:
http://cdn.dzone.com/sites/all/files/refcardz/rc112-010d-uml.pdf

The book UML Distilled:
http://www.amazon.com/UML-Distilled-Standard-Modeling-Language/dp/0321193687