JAVA PROGRAMMING COURSE

OBJECT CONVERSION IN JAVA

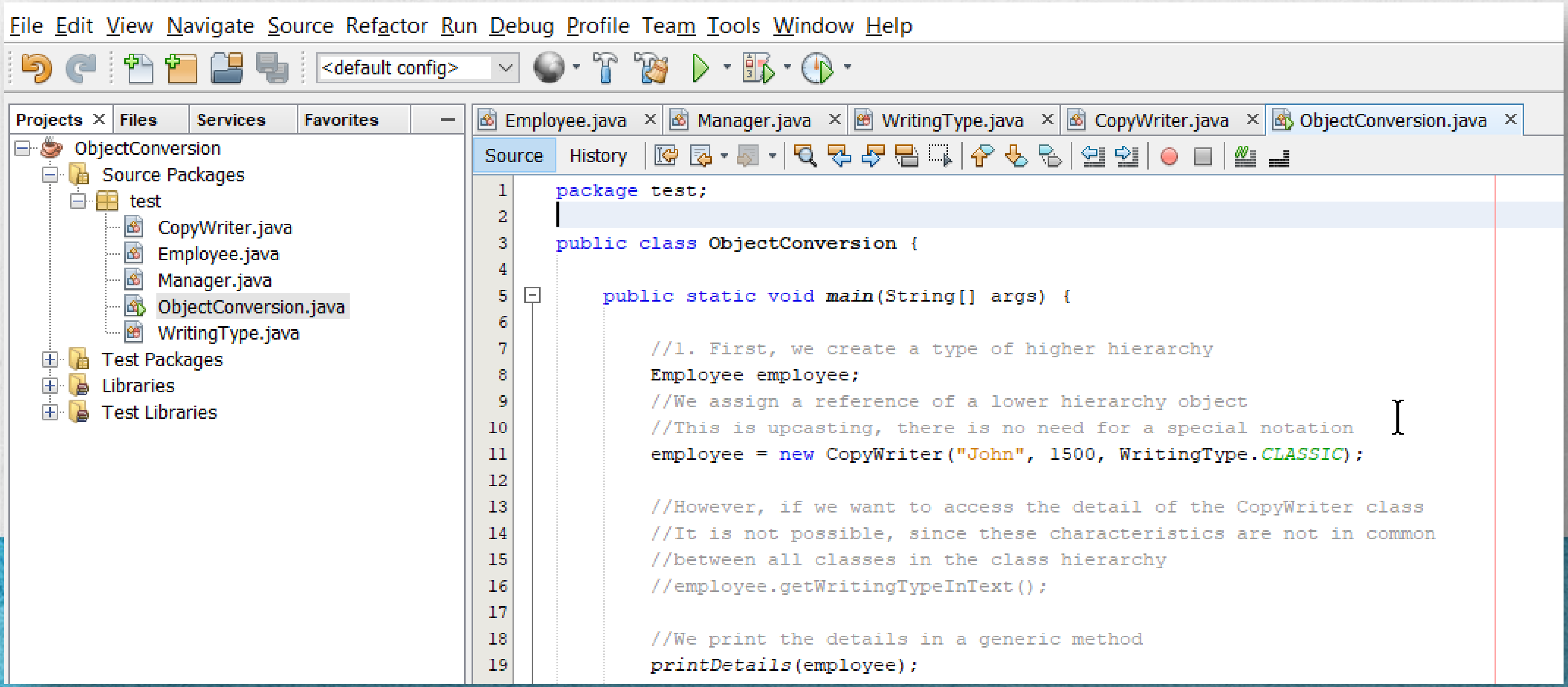
By the expert: Ubaldo Acosta

JAVA PROGRAMMING COURSE

www.globalmentoring.com.mx

EXERCISE OBJECTIVE

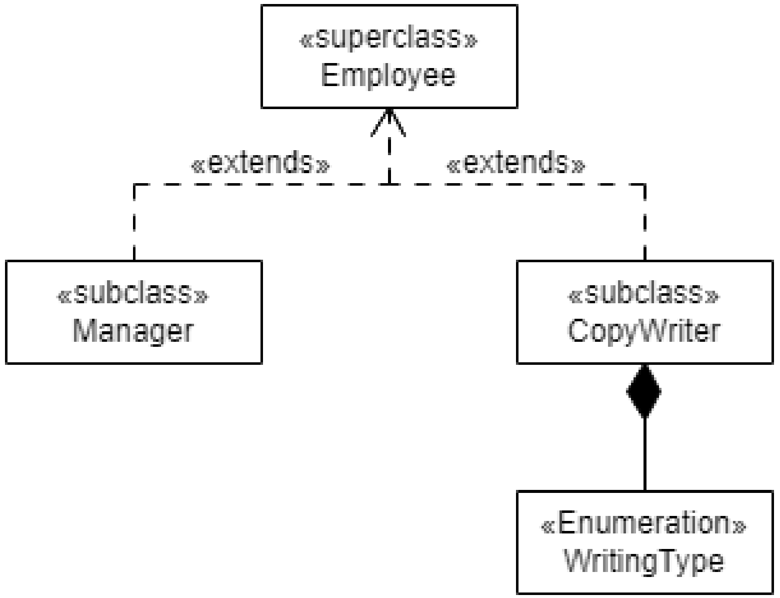
Put into practice the concept of conversion of objects in Java. At the end we should observe the following:



```java
package test;

public class ObjectConversion {

    public static void main(String[] args) {

        //1. First, we create a type of higher hierarchy
        Employee employee;
        //We assign a reference of a lower hierarchy object
        //This is upcasting, there is no need for a special notation
        employee = new CopyWriter("John", 1500, WritingType.CLASSIC);

        //However, if we want to access the detail of the CopyWriter class
        //It is not possible, since these characteristics are not in common
        //between all classes in the class hierarchy
        //employee.getWritingTypeInText();

        //We print the details in a generic method
        printDetails(employee);
```

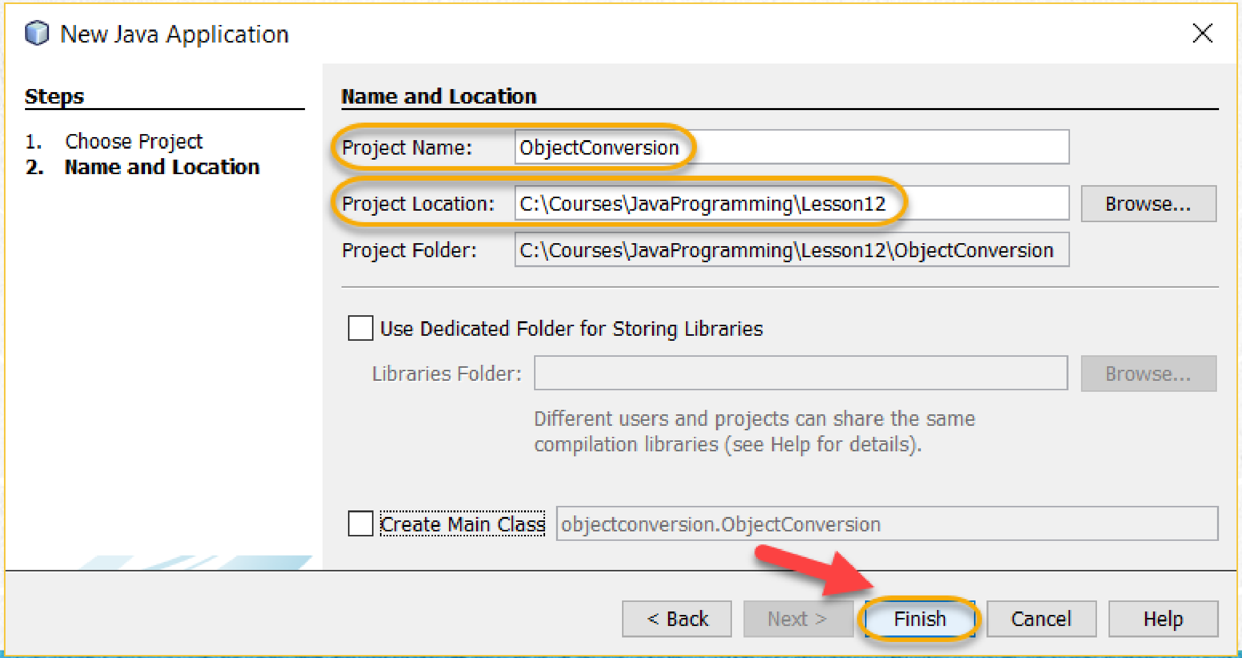
UML DIAGRAM OF THE EXERCISE

This is the UML diagram of the exercise :



1. CREATE A NEW PROJECT

Create a new project:



New Java Application

Steps

1. Choose Project
2. **Name and Location**

Name and Location

Project Name: ObjectConversion

Project Location: C:\Courses\JavaProgramming\Lesson12

Project Folder: C:\Courses\JavaProgramming\Lesson12\ObjectConversion

[ ] Use Dedicated Folder for Storing Libraries

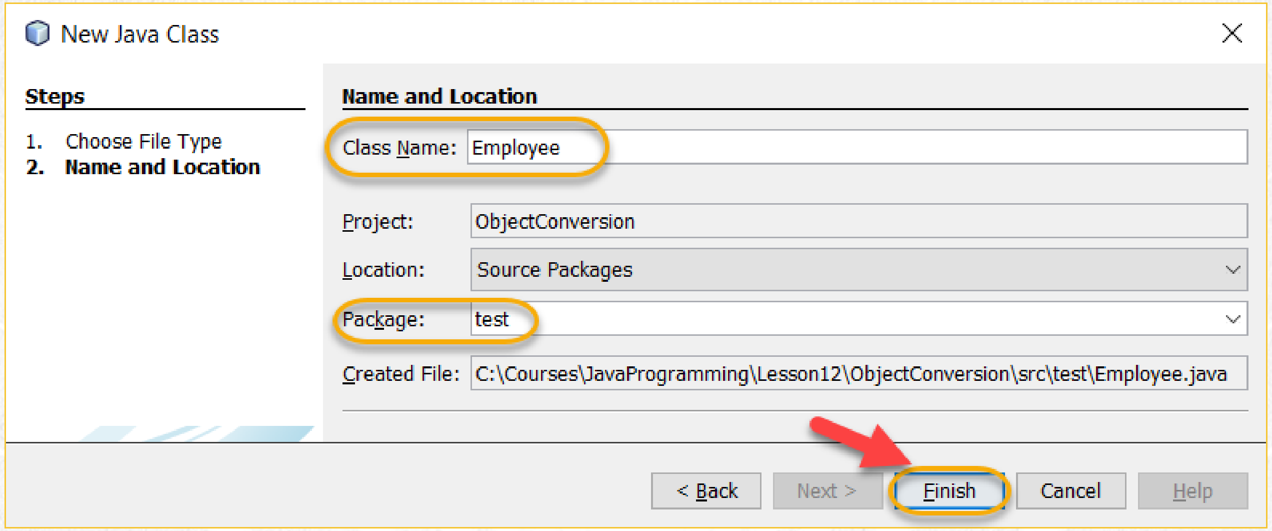
Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

[ ] Create Main Class objectconversion.ObjectConversion

2. CREATE A NEW CLASS

Create a new class:



Class Name: Employee

Project: ObjectConversion

Location: Source Packages

Package: test

Created File: C:\Courses\JavaProgramming\Lesson12\ObjectConversion\src\test\Employee.java

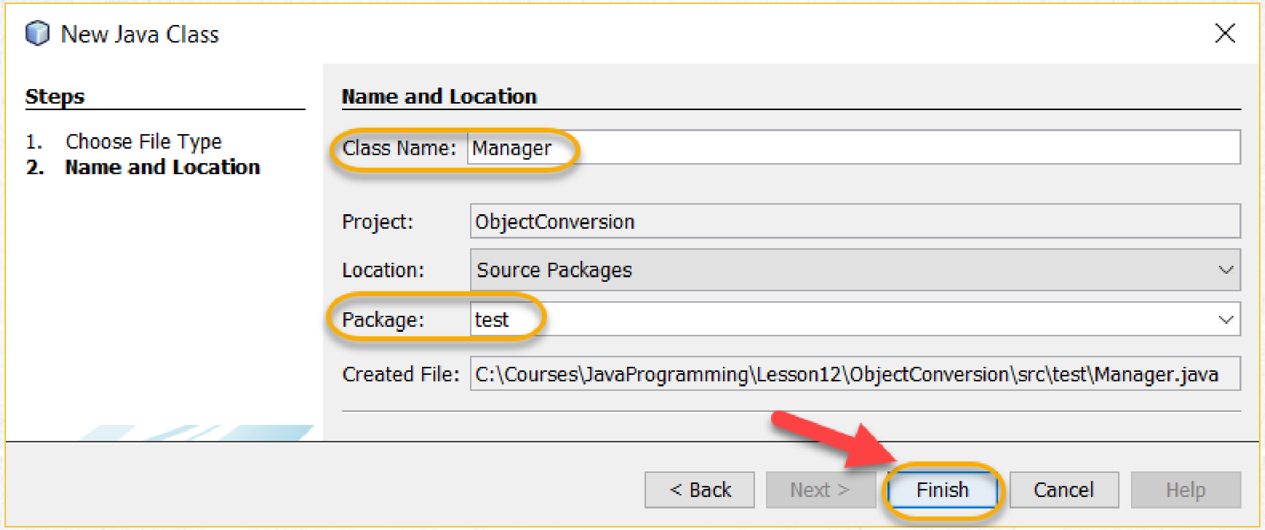
3. MODIFY THE CODE

Employee.java:

```
package test;

public class Employee {

    protected String name;
    protected double salary;

    protected Employee(String name, double salary){
        this.name = name;
        this.salary = salary;
    }

    public String getDetails(){
        return "Name: " + name + ", salary: " + salary;
    }

      public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }
 }
```

4. CREATE A NEW CLASS

Create a new class:



New Java Class

Steps

1. Choose File Type
2. **Name and Location**

Name and Location

Class Name:	Manager
Project:	ObjectConversion
Location:	Source Packages
Package:	test
Created File:	C:\Courses\JavaProgramming\Lesson12\ObjectConversion\src\test\Manager.java

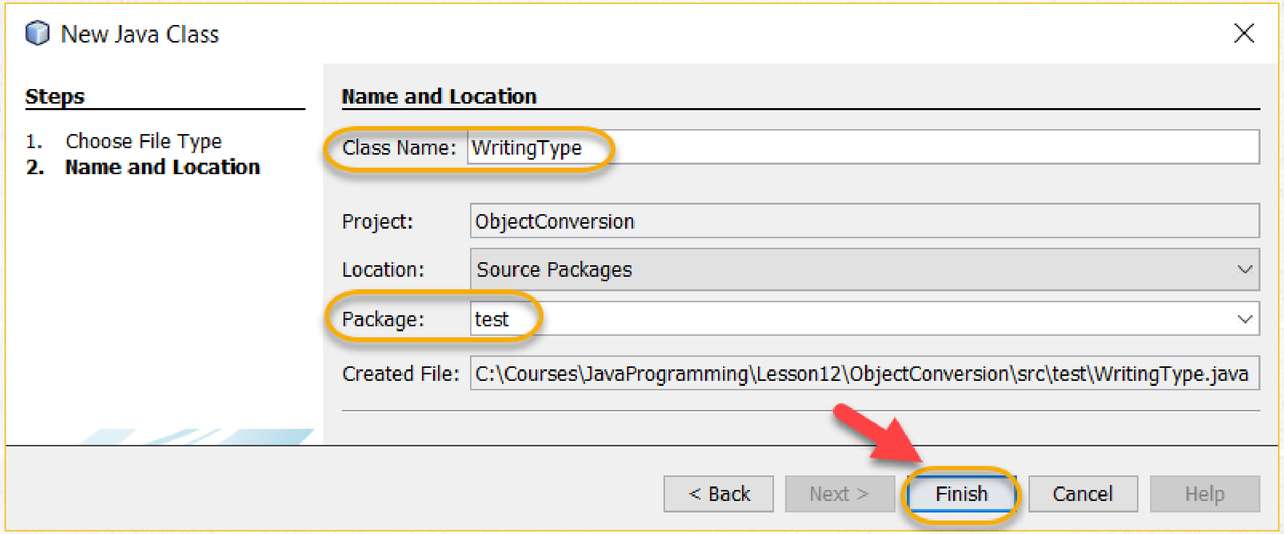
5. MODIFY THE CODE

Manager.java:

```java
package test;

public class Manager extends Employee{

    private String department;

    public Manager(String name, double salary, String department ){
        super(name, salary);
        this.department = department;
    }

    //overwrite the inherited parent method
    public String getDetails(){
        //In order not to repeat code, we can use
        //the parent method and only add the child attribute
        return super.getDetails()+ ", department: " + department;
    }

     public String getDepartment() {
        return department;
    }

    public void setDepartment(String department) {
        this.department = department;
    }
}
```

6. CREATE A NEW CLASS

Create a new class:



New Java Class

Steps

1. Choose File Type
2. **Name and Location**

Name and Location

Class Name: WritingType

Project: ObjectConversion

Location: Source Packages

Package: test

Created File: C:\Courses\JavaProgramming\Lesson12\ObjectConversion\src\test\WritingType.java

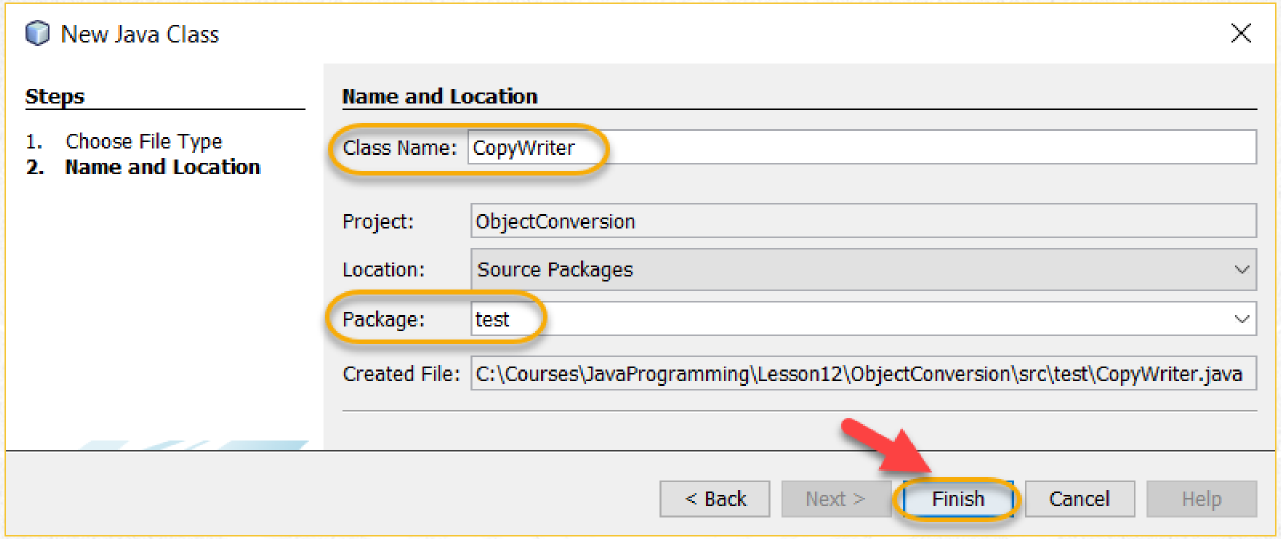
7. MODIFY THE CODE

WritingType.java:

```
package test;

public enum WritingType {

    CLASSIC("Writing by hand"),
    MODERN("Digital writing");

    private final String description;

    private WritingType(String description) {
        this.description = description;
    }

    public String getDescription() {
        return description;
    }
}
```

8. CREATE A NEW CLASS

Create a new class:


New Java Class

Steps

1. Choose File Type
2. **Name and Location**

Name and Location

Class Name: CopyWriter

Project: ObjectConversion

Location: Source Packages

Package: test

Created File: C:\Courses\JavaProgramming\Lesson12\ObjectConversion\src\test\CopyWriter.java

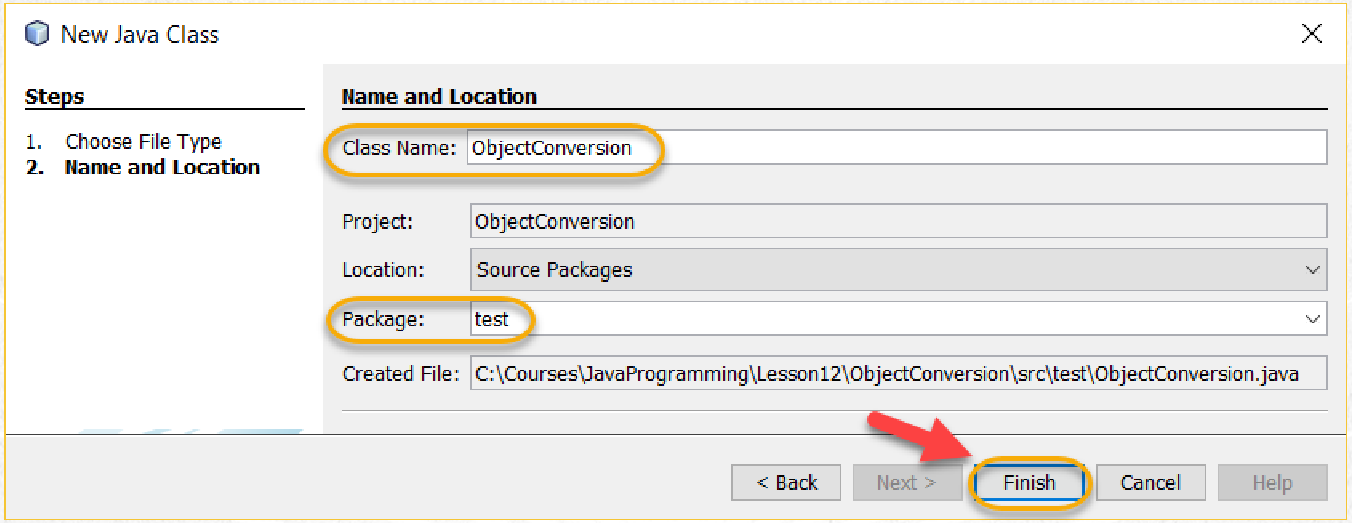
9. MODIFY THE CODE

CopyWriter.java:

```java
package test;

public class CopyWriter extends Employee{

    //We can use an enumeration for writing type options
    final WritingType writingType;

    public CopyWriter(String name, double salary, WritingType writingType) {
        super(name, salary);
        this.writingType = writingType;
    }

    public String getDetails(){
        //In order not to repeat code, we can use
        //the parent method and only add the child attribute
        return super.getDetails()+ ", writingType: " + writingType.getDescription();
    }

    public WritingType getWritingType() {
        return writingType;
    }

    public String getWritingTypeInText() {
        return writingType.getDescription();
    }
}
```

10. CREATE A NEW CLASS

Create a new class:



Name and Location

Class Name: ObjectConversion

Project: ObjectConversion

Location: Source Packages

Package: test

Created File: C:\Courses\JavaProgramming\Lesson12\ObjectConversion\src\test\ObjectConversion.java

11. MODIFY THE CODE

ObjectConversion.java:

```java
package test;

public class ObjectConversion {

    public static void main(String[] args) {

        //1. First, we create a type of higher hierarchy
        Employee employee;
        //We assign a reference of a lower hierarchy object
        //This is upcasting, there is no need for a special notation
        employee = new CopyWriter("John", 1500, WritingType.CLASSIC);

        //However, if we want to access the detail of the CopyWriter class
        //It is not possible, since these characteristics are not in common
        //between all classes in the class hierarchy
        //employee.getWritingTypeInText();

        //We print the details in a generic method
        printDetails(employee);

        //2. We can do the same with the Manager class
        //This is upcasting, so it does not require special syntax
        employee = new Manager("Katty", 1800, "Systems");

        //But if we want direct access to the employee variable
        //the detail of the Manager object is not possible
        //employee.getDepartment();
```

11. MODIFY THE CODE

ObjectConversion.java:

```
        //We use the same method to print the details
        printDetails(employee);
    }

    //Generic method to print the details of the Employee hierarchy
    private static void printDetails(Employee employee) {

        String result = null;

        //Print details is general for everyone, since it is polymorph
        //no conversion is needed, we can call the method directly
        System.out.println("\nDetails: " + employee.getDetails());

        //But the details of each class we must do a downcasting
        if (employee instanceof CopyWriter) {
            //We convert the object to the desired lower type
            //Convert object - Downcasting
            CopyWriter writer = (CopyWriter) employee;
            //Finally we can access the methods of the CopyWriter class
            result = writer.getWritingTypeInText();

            //Another way is to make the conversion in the same line of code.
            //This is very compun to find in Java
            //to avoid the creation of unnecessary variables
            result = ((CopyWriter) employee).writingType.getDescription();

            System.out.println("typeWriting result:" + result);
```

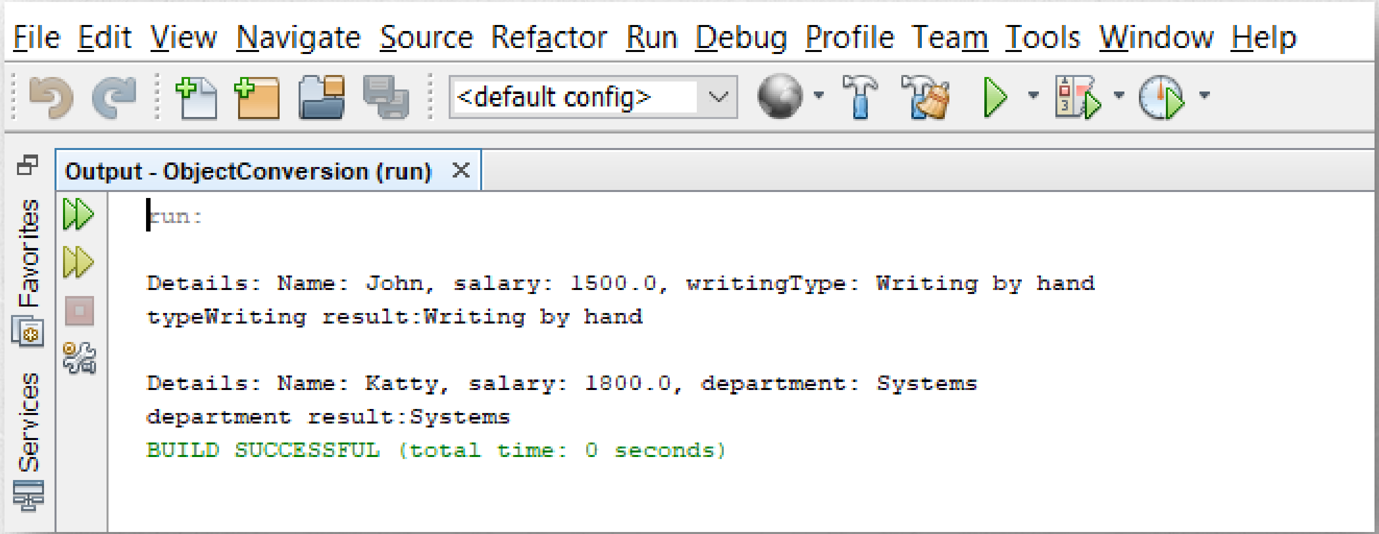
11. MODIFY THE CODE

ObjectConversion.java:

```
        } else if (employee instanceof Manager) {
            //We do the downcasting in the same line of code
            //we saved a variable
            result = ((Manager) employee).getDepartment();
            System.out.println("department result:" + result);
        }
    }
}
```

12. EXECUTE THE PROJECT

The result is as follows:


File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Output - ObjectConversion (run)
Favorites
Services

```
run:

Details: Name: John, salary: 1500.0, writingType: Writing by hand
typeWriting result:Writing by hand

Details: Name: Katty, salary: 1800.0, department: Systems
department result:Systems
BUILD SUCCESSFUL (total time: 0 seconds)
```

EXERCISE CONCLUSION

With this exercise we have put into practice the concept of object conversion, also known as casting.

We have seen that the upcasting does not need a particular syntax, but when doing a downcasting, the compiler requires us to confirm if we really want to do that conversion, besides that conversion must be within the hierarchy of classes that we have defined.

ONLINE COURSE

JAVA PROGRAMMING

By: Eng. Ubaldo Acosta

Global Mentoring