# Practices for Lesson 4: Overriding Methods and Applying Polymorphism

**Chapter 4**

# Practices for Lesson 4

## Practices Overview

In these practices, you will

- Use static method
- Override methods, including the `toString` method in the Object class
- Create a method in a class that uses the `instanceof` operator to determine which object was passed to the method
- Overload methods
- Use casting

# Practice 4-1: Summary Level: Overriding and Overloading Methods

## Overview

In this practice, you will use a static method, override the `toString` method of the `Object` class in the `Employee` class and in the `Manager` class. You will create an `EmployeeStockPlan` class with a `grantStock` method that uses the `instanceof` operator to determine how much stock to grant based on the employee type.

## Assumptions

## Tasks

1. Open the `Employee04-01Prac` project in the `practices/practice1` directory.
2. Edit the `Employee` class:

   a. Delete the instance method `printEmployee()`.

   b. Override the `toString()` method from the `Object` class. `Object`'s `toString` method returns a `String`.

      I. Add a `return` statement that returns a string that includes the employee ID, name, Social Security number, and a salary as a formatted string, with each line separated with a newline character (`"\n"`).

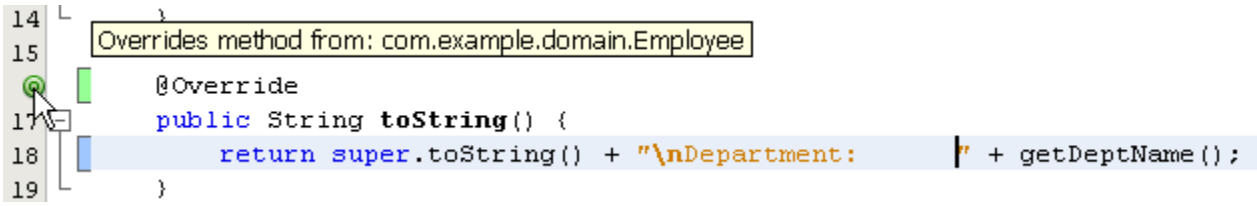
      II. To format the double salary, use the following:

      ```
      i.NumberFormat.getCurrencyInstance().format(getSalary())
      ```

      III. Fix any missing import statements.

      IV. Save the class.
3. Override the `toString()` method in the `Manager` class to include the `deptName` field value. Separate this string from the Employee string with a newline character.

   Note the Green circle icon with the "o" in the center beside the method signature in the `Manager` class. This indicates that NetBeans is aware that this method overrides the method from the parent class, Employee. Hold the cursor over the icon to read what this icon represents:

   Click the icon, and NetBeans will open the `Employee` class and position the view to the `toString()` method.
4. (Optional) Override the `toString()` method in the `Director` class as well, to display all the fields of a Director and the available budget.

5. Create a new class called `EmployeeStockPlan` in the package `com.example.business`. This class will include a single method, `grantStock`, which takes an `Employee` object as a parameter and returns an integer number of stock options based on the employee type:

| Employee Type | Number of Stock Options |
|---|---|
| Director | 1000 |
| Manager | 100 |
| All other Employees | 10 |

   a. Add a `grantStock` method that takes an `Employee` object reference as a parameter and returns an integer
   b. In the method body, determine what employee type was passed in using the `instanceof` keyword and return the appropriate number of stock options based on that type.
   c. Resolve any missing import statements.
   d. Save the `EmployeeStockPlan` class.

6. Modify the `EmployeeTest` class:

   a. Add a static `printEmployee` method that invokes the toString method of the Employee class.

```
public static void printEmployee(Employee emp) {

        System.out.println(emp);
    }
```

   b. Overload the `printEmployee` method to take a second parameter, `EmployeeStockPlan`, and print out the number of stock options that this employee will receive.

      i. The new `printEmployee` method should call the first `printEmployee` method and the number of stocks granted to this employee:

```
printEmployee (emp);
System.out.println("Stock Options:   " + esp.grantStock(emp));
```

   c. Above the `printEmployee` method calls in the `main` method, create an instance of the `EmployeeStockPlan` and pass that instance to each of the `printEmployee` methods:

```
EmployeeStockPlan esp = new EmployeeStockPlan();
printEmployee(eng, esp);
```

   d. Modify the remaining `printEmployee` invocations.

```
printEmployee(adm, esp);
printEmployee(mgr, esp);
printEmployee(dir, esp);
```

e. Modify the code used to display the Managers stock plan after invoking the `raiseSalary` method to

```
printEmployee(mgr, esp);
```

7. Save the `EmployeeTest` class and run the application. You should see output for each employee that includes the number of Stock Options, such as:

```
Employee id:         101
Employee name:       Jane Smith
Employee SSN:  012-34-5678
Employee salary:     $120,345.27
Stock Options:       10
```

8. It would be nice to know what type of employee each employee is. Add the following to your original `printEmployee` method above the print statement that prints the employee data fields:

```
System.out.println("Employee type:       " +
emp.getClass().getSimpleName());
```

This will print out the simple name of the class (`Manager`, `Engineer`, and so on). The output of the first employee record should now look like this:

```
Employee type:       Engineer
Employee id:         101
Employee name:       Jane Smith
Employee SSN:  012-34-5678
Employee salary:     $120,345.27
Stock Options:       10
```

## Practice 4-1: Detailed Level: Overriding and Overloading Methods

### Overview

In this practice, you will use a static method, override the `toString` method of the `Object` class in the `Employee` class and in the `Manager` class. You will create an `EmployeeStockPlan` class with a `grantStock` method that uses the `instanceof` operator to determine how much stock to grant based on the employee type.

**Tasks**

1. Open the `Employee04-01Prac` project in the `practices` directory.
   a. Select File > Open Project.
   b. Browse to `/home/oracle/labs/04-Polymorphism/practices/practice1`.
   c. Select `Employee04-01Prac` and click Open Project.
2. Edit the `Employee` class: to override the `toString()` method from the `Object` class. `Object`'s `toString` method returns a `String`.
   a. Delete the instance method `printEmployee()` from the `Employee` class.

```
public  void printEmployee() {

        System.out.println(); // Print a blank line as a
separator
        // Print out the data in this Employee object
        System.out.println("Employee id:         " +
getEmpId());
        System.out.println("Employee name:       " + getName());
        System.out.println("Employee SSN:  " + getSsn());
        System.out.println("Employee salary:     " +
NumberFormat.getCurrencyInstance().format((double)
getSalary()));
    }
```

   b. Add the `toString` method to the `Employee` class with the following signature:
      `public String toString() {`
   c. Add a `return` statement that returns a string that includes the employee information: ID, name, Social Security number, and a formatted salary like this:

```
return "Employee ID:     " + getEmpId() + "\n" +
       "Employee Name:   " + getName() + "\n" +
       "Employee SSN:    " + getSsn() + "\n" +
       "Employee Salary: " +
NumberFormat.getCurrencyInstance().format(getSalary());
```

   d. Save the `Employee` class.
3. Override the `toString` method in the `Manager` class to include the `deptName` field value.
   a. Open the `Manager` class.

b. Add a `toString` method with the same signature as the `Employee toString` method:

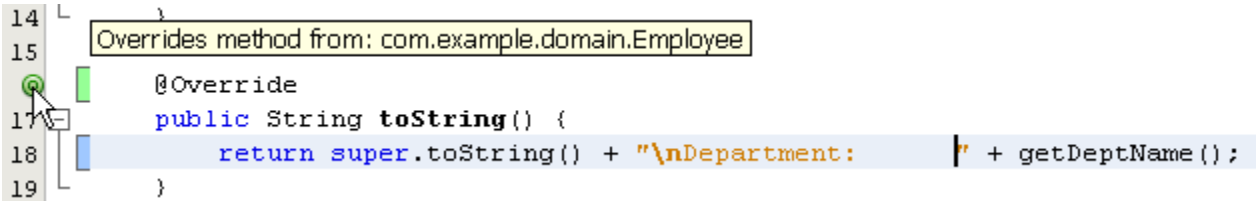
```
public String toString() {
```

The `toString` method in the `Manager` class overrides the `toString` method inherited from the `Employee` class.

c. Call the parent class method by using the `super` keyword and add the department name:

```
return super.toString() + "\nDepartment:      " + getDeptName();
```

Note the Green circle icon with the “o” in the center beside the method signature in the `Manager` class. This indicates that NetBeans is aware that this method overrides the method from the parent class, Employee. Hold the cursor over the icon to read what this icon represents:




Click the icon, and NetBeans will open the `Employee` class and position the view to the `toString()` method.

d. Save the `Manager` class.

4. (Optional) Override the `toString` method in the `Director` class as well, to display all the fields of a director and the available budget.

5. Create a new class called `EmployeeStockPlan` in the package `com.example.business`. This class will include a single method, `grantStock`, which takes an `Employee` object as a parameter and returns an integer number of stock options based on the employee type:

| Employee Type | Number of Stock Options |
|---|---|
| Director | 1000 |
| Manager | 100 |
| All other Employees | 10 |

a. Create the new package and class in one step by right-clicking Source Package, and then selecting New > Java Class.

b. In the New Java Class window, perform the following steps:

1) Enter the class name as `EmployeeStockPlan`.

2) Enter the package name as `com.example.business`.

3) Click Finish.

c. Add fields to the `EmployeeStockPlan` class to define the stock levels, like this:

```
    private final int employeeShares = 10;
    private final int managerShares = 100;
    private final int directorShares = 1000;
```

d. Add a `grantStock` method that takes an `Employee` object reference as a parameter and returns an integer:

```
    public int grantStock(Employee emp) {
```

e. In the method body, determine what employee type was passed in using the `instanceof` keyword and return the appropriate number of stock options based on that type. Your code might look like this:

```
        // Stock is granted based on the employee type
        if (emp instanceof Director) {
            return directorShares;
        } else {
            if (emp instanceof Manager) {
                return managerShares;
            } else {
                return employeeShares;
            }
        }
```

f. Resolve any missing import statements.

g. Save the `EmployeeStockPlan` class.

6. Modify the `EmployeeTest` class:

a. Add a static `printEmployee` method.

```
public static void printEmployee(Employee emp) {

        System.out.println(emp);
    }
```

**Note:** This code of line invokes the `toString()` method of the Employee class.

The instance method `printEmployee` has been converted to a static method in this practice.

b. Overload the `printEmployee` method to take a second parameter, `EmployeeStockPlan`, and print out the number of stock options that this employee will receive.

I. Create another `printEmployee` method that takes an instance of the `EmployeeStockPlan` class:

a. `public static void printEmployee(Employee emp, EmployeeStockPlan esp) {`

II. This method first calls the original `printEmployee` method:

a. `printEmployee(emp);`

III. Add a print statement to print out the number of stock options that the employee is entitled to:

```
System.out.println("Stock Options:       " +
esp.grantStock(emp));
```

c. Resolve any missing import statements.

d. Above the `printEmployee` method calls in the main method, create an instance of the `EmployeeStockPlan` and pass that instance to each of the `printEmployee` methods:

```
EmployeeStockPlan esp = new EmployeeStockPlan();
printEmployee(eng, esp);
```

e. Modify the remaining `printEmployee` invocations.

```
printEmployee(adm, esp);
printEmployee(mgr, esp);
printEmployee(dir, esp);
```

f. Modify the code used to display the Managers stock plan after invoking the `raiseSalary` method to

```
printEmployee(mgr, esp);
```

8. Save the `EmployeeTest` class and run the application. You should see output for each employee that includes the number of Stock Options, such as:

```
Employee id:         101
Employee name:       Jane Smith
Employee SSN:  012-34-5678
Employee salary:     $120,345.27
Stock Options:       10
```

9. It would be nice to know what type of employee each employee is. Add the following to your original `printEmployee` method above the print statement that prints the employee data fields:

```
System.out.println("Employee type:       " +
emp.getClass().getSimpleName());
```

This will print out the simple name of the class (`Manager`, `Engineer`, etc). The output of the first employee record should now look like this:

```
Employee type:       Engineer
Employee id:         101
Employee name:       Jane Smith
Employee SSN:  012-34-5678
Employee salary:     $120,345.27
Stock Options:       10
```

## Practice 4-2: Summary Level: Using Casting

### Overview

In this practice, you will cast object references and invoke appropriate methods.

You are provided with an `Employee04-02Prac` project that has some compilation errors. You will fix the errors and review the desired output. On running the project, you will encounter a runtime exception for which you need to determine the cause and fix it.

### Tasks

Open the `Employee04-02Prac` project in the `practices/practice2` directory.

1. Examine the `main` method of `EmployeeTest.java` and identify lines of code that does object casting.
2. Examine the compilation errors related to casting and identify their cause.
3. Fix the compilation errors.
4. Run the project. Verify if you get a run time exception.
5. Identify the specific exception and determine the line number that caused the run time exception.
   a. Fix the cause of the exception.
6. Run the project and verify the output.

# Practice 4-2: Detailed Level: Using Casting

## Overview

In this practice, you will cast object references and invoke appropriate methods.

You are provided with the `Employee04-02Prac` project that has some compilation errors. You will fix the errors and review the desired output. On running the project, you will encounter a runtime exception for which you need to determine the cause and fix it.

## Tasks

1. Open the `Employee04-02Prac` project in the `/home/oracle/labs/04-Polymorphism/practices/practice2` directory.
2. Examine the `main` method of `EmployeeTest.java` and identify lines of code that does object casting.
3. Examine the compilation errors at line numbers 17, 20, and 23 related to casting and identify their cause.

```
12          // Create the classes as per the practice
13          Engineer eng = new Engineer(101, "Jane Smith", "012-34-5678", 120_345.27);
14          Employee emp = new Employee(13, "Lionel Power", "099-90-6789", 67_990.90);
15          Employee obj = new Engineer(102, "Robert Stock", "012-54-7812", 220_345.27);
16
            obj.engineerMethod();
18          printEmployee(obj);
19
            Engineer engobj = new Employee(1, "Brenda Wills", "013-78-5678", 221_500.00);
21          printEmployee(engobj);
22
            String s = (String) emp;
24
25      }
```

4. Fix the compilation errors.
   a. Modify line 17 to: `eng.engineerMethod();`
   b. Modify line 20 to downcast:
      `Engineer engobj = (Engineer)new Employee(1, "Brenda Wills", "013-78-5678", 221_500.00);`
   c. Comment out line 23: `//String s = (String) emp;`
5. On the Projects tab, select `Employee04-02Prac`, right-click and select **Run** from the drop down menu.
   a. Verify if you get a run time exception:

```
Output - EmployeeSolution (run)
run:
Method specific to Engineer class

Employee id:          102
Employee name:        Robert Stock
Employee Soc Sec #:   012-54-7812
Employee salary:      $220,345.27
Exception in thread "main" java.lang.ClassCastException: com.example.domain.Employee cannot be cast to com.example.domain.Engineer
        at com.example.EmployeeTest.main(EmployeeTest.java:33)
Java Result: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

6. Identify the specific exception and determine the cause of the run time exception.

a. Fix the cause of the exception.
   Modify Line 20 to:
   ```
   Engineer engobj =  new Engineer(1, "Brenda Wills", "013-78-5678", 221_500.00);
   ```

7. Run the project and verify the output.

```
run:
Method specific to Engineer class

Employee id:          102
Employee name:        Robert Stock
Employee Soc Sec #:   012-54-7812
Employee salary:      $220,345.27

Employee id:          1
Employee name:        Brenda Wills
Employee Soc Sec #:   013-78-5678
Employee salary:      $221,500.00
BUILD SUCCESSFUL (total time: 0 seconds)
```

## Practice 4-3: Summary Level: Applying the Singleton Design Pattern

### Overview

In this practice, you will take an existing application and refactor the code to implement the Singleton design pattern.

### Summary

You are working on server software that synchronizes with other servers. Your task is to create a Singleton class which stores the hostnames of the servers to connect with. The server list is declared in a static initialization block.

### Tasks

1. Open the `Singleton04-03Prac` project.
   a. Select File > Open Project.
   b. Browse to `\home\oracle\labs\04-Polymorphism\practices\practice3`.
   c. Select `Singleton04-03Prac` and click Open Project.
2. Expand the project directories.
3. Modify the `PeerSingleton` class to implement the Singleton design pattern.
   a. Open the `PeerSingleton.java` file (under the `com.example` package).
   b. Change the constructor's access level to `private`.
   c. Add a new field named `instance`. The field should be:
      i. `private`
      ii. Marked `static`
      iii. Marked `final`
      iv. Type of `PeerSingleton`
      v. Initialized to a new `PeerSingleton` instance
   d. Create a static method named `getInstance` that returns the value stored in the `instance` field.
4. Modify the `Main` class to use the singleton.
   a. Open the `Main.java` file (under the `com.example` package).
   b. Perform the following steps in the `main` method:
      1) Create a `PeerSingleton` reference named `peerList01` and initialize it using the `getInstance` method.
      2) Create a second `PeerSingleton` reference named `peerList02` and initialize it using the `getInstance` method.
      3) Display the host names by invoking `getHostNames` on `peerList01` in a `for` loop.
      4) Next, display the host names by invoking `getHostNames` on `peerList02` in a `for` loop.
5. Run the project. You should see a list of host names.

## Practice 4-3: Detailed Level: Applying the Singleton Design Pattern

### Overview

In this practice, you will take an existing application and refactor the code to implement the Singleton design pattern.

### Summary

You are working on server software that synchronizes with other servers. Your task is to create a Singleton class, which stores the hostnames of the servers to connect with. The server list is declared in a static initialization block.

### Tasks

1. Open the `Singleton04-03Prac` project.
   a. Select File > Open Project.
   b. Browse to `\home\oracle\labs\04-Polymorphism\practices\practice3`.
   c. Select `Singleton04-03Prac` and click Open Project.
2. Expand the project directories.
3. Modify the `PeerSingleton` class to implement the Singleton design pattern.
   a. Open the `PeerSingleton.java` file (under the `com.example` package).
   b. Change the constructor's access level to `private`.

```
private PeerSingleton()
{
 }
```

4. Add a new field named `instance`. The field should be:
   a. `private`
   b. Marked `static`
   c. Marked `final`
   d. Type of `PeerSingleton`
   e. Initialized to a new `PeerSingleton` instance

```
private static final PeerSingleton instance = new
PeerSingleton();
```

   f. Create a static method named `getInstance` that returns the value stored in the `instance` field.

```
public static PeerSingleton getInstance() {
    return instance;
  }
```

5. Modify the `Main` class to use the singleton.
   a. Open the `Main.java` file (under the `com.example` package).
   b. Perform the following steps in the `main` method:
   c. Create a `PeerSingleton` reference named `peerList01` and initialize it using the `getInstance` method.

```
PeerSingleton peerList01 = PeerSingleton.getInstance();
```

d. Create a second `PeerSingleton` reference named `peerList02` and initialize it using the `getInstance` method.

```
PeerSingleton peerList02 = PeerSingleton.getInstance();
```

e. Display the host names by invoking `getHostNames` on `peerList01` in a `for` loop.

```
for(String hostName:peerList01.getHostNames()){
      System.out.println("Host name: " + hostName);
    }
```

f. Next, display the host names by invoking `getHostNames` on `peerList02` in a `for` loop.

```
System.out.println();
    for(String hostName:peerList02.getHostNames()){
      System.out.println("Host name: " + hostName);
    }
```

6. Run the project. You should see a list of host names.