Μία εταιρεία πληρώνει τους εργαζόμενους της σε εβδομαδιαία βάση. Οι αποδοχές των εργαζόμενων υπολογίζονται με τέσσερις διαφορετικούς τρόπους :

Α) Μισθωτοί **(SalariedEmployee)**, πληρώνονται με σταθερό ποσό εβδομαδιαία ανεξάρτητα από το πόσες ώρες δούλεψαν.

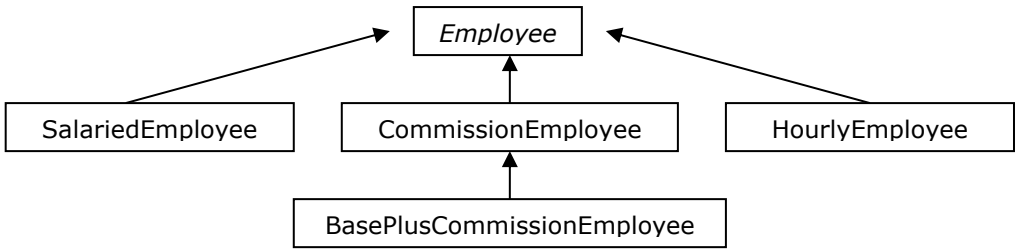
Β) Ωρομίσθιοι **(HourlyEmployee)**, πληρώνονται με την ώρα και λαμβάνουν επιπλέον αποδοχές για τις υπερωρίες.

Γ) Πωλητές **(CommissionEmployee)**, πληρώνονται με ποσοστά επί των πωλήσεων.

Δ) Μισθωτοί πωλητές **(BasePlusCommissionEmployee)**, πληρώνονται με ένα βασικό ποσό και επιπλέον με ποσοστό επί των πωλήσεων.

Η εταιρεία ζητά να αναπτύξετε μια Java εφαρμογή **(TestPayroll)** για να υπολογίζει την μισθοδοσία των υπαλλήλων της. Οι κλάσεις Employee, SalariedEmployee, HourlyEmployee, CommissionEmployee, BasePlusCommissionEmployee να ανήκουν στο πακέτο payroll και στην εφαρμογή **TestPayroll** να γίνει import του πακέτου payroll.

<u>Ανάλυση άσκησης</u>



Χρησιμοποιούμε την «γενική» κλάση **Employee** για να περιγράψουμε τις βασικές ιδιότητες και μεθόδους που είναι κοινές για όλες τις κατηγορίες εργαζομένων. Το διάγραμμα δείχνει πως οι κλάσεις για τις διάφορες κατηγορίες εργαζομένων κληρονομούν την κλάση του εργαζομένου. Στο παράδειγμα μας δεν θα υπάρξει η ανάγκη να δημιουργηθεί αντικείμενο τύπου Employee, θα πρέπει κάθε φορά να καθορίζουμε τον τύπο του εργαζομένου, για το λόγο αυτό η κλάση Employee θα είναι **abstract** (Μια τέτοια τάξη δεν μπορεί να χρησιμοποιηθεί για τη δημιουργία αντικειμένων). Τα χαρακτηριστικά για την τάξη Employee είναι firstName, lastName, socialSecurityNumber. Επίσης θα πρέπει να υπάρχει μια μέθοδος για τον υπολογισμό των αποδοχών που θα την ονομάσουμε earnings. Η μέθοδος αυτή απαιτείται για

όλους τους εργαζομένους αλλά υπολογίζει διαφορετικά τις αποδοχές ανά τύπο εργαζομένου. Για το λόγο αυτό θα δηλωθεί ως **abstract** στην υπερτάξη Employee και η κάθε υποτάξη θα παρακάμπτει την μέθοδο **earnings** έχοντας την κατάλληλη υλοποίηση (method overriding).

Η κλάση **SalariedEmployee** κληρονομεί και επεκτείνει την κλάση Employee. Η κλάση περιλαμβάνει έναν κατασκευαστή που με την εντολή `super(first,last,socialSecurityNumber);` καλεί τον κατασκευαστή της υπερτάξης για να αρχικοποιήσει τα πεδία `first,last,socialSecurityNumber` τα οποία κληρονομεί από την υπερτάξη. Με την μέθοδο `earnings` υπολογίζουμε τις αποδοχές του Μισθωτού υπαλλήλου. Η μέθοδος `toString()` της υπερτάξης Employee επιστρέφει «όνομα, επίθετο, Αριθμό ασφάλισης». Η μέθοδος `toString()` της υποτάξης `SalariedEmployee` «παρακάμπτει» την μέθοδο της υπερτάξης και επιστρέφει **τύπο εργαζομένου,** όνομα, επίθετο, Αριθμό ασφάλισης». Κατά κάποιο τρόπο, η νέα μέθοδος καταργεί τη δράση της αρχικής. Επιπλέον καλεί την μέθοδο `toString()` της υπερτάξης `Employee` με την χρήση του super για να λάβει τις πληροφορίες για τα «όνομα, επίθετο, Αριθμό ασφάλισης».

Η κλάση **HourlyEmployee** κληρονομεί και επεκτείνει την κλάση Employee. Η κλάση όμοια με την SalariedEmployee περιλαμβάνει έναν κατασκευαστή που με την εντολή `super(first,last,socialSecurityNumber );` καλεί τον κατασκευαστή της υπερτάξης για να αρχικοποιήσει τα πεδία `first,last,socialSecurityNumber` τα οποία κληρονομεί από την υπερτάξη.

Η κλάση **BasePlusCommissionEmployee** κληρονομεί και επεκτείνει την κλάση CommissionEmployee και συνεπώς και την κλάση Employee. Η κλάση περιλαμβάνει έναν κατασκευαστή που με την εντολή `super( first, last, socialSecurityNumber, grossSalesAmount, rate );`
καλεί τον κατασκευαστή της υπερτάξης που είναι η CommissionEmployee και να αρχικοποιήσει τα πεδία που κληρονομεί. Στη μέθοδο earnings υπολογίζει τον μισθό με την εντολή `return getBaseSalary() + super.earnings();`
Αυτό είναι ένα παράδειγμα επαναχρησιμοποιήσιμου κώδικα που επιτυγχάνετε με την κληρονομικότητα.

Στο πρόγραμμα **Payroll** έχουμε έναν πίνακα με 4 εργαζόμενους. Στη συνέχεια δημιουργούμε έναν εργαζόμενο για κάθε τύπο και τους αποθηκεύουμε στον πίνακα. Στο for loop εμφανίζουμε τα στοιχεία του κάθε εργαζομενου του πίνακα. Με την γραμμή

`if ( employees[ i ] instanceof BasePlusCommissionEmployee )` συμπεραίνουμε εάν το αντικείμενο έχει δημιουργηθεί από την κλάση `BasePlusCommissionEmployee`, εάν ναι τύπωσε τον παλιό βασικό μισθό, υπολόγισε και τύπωσε αύξηση 10% επι του βασικού μισθού και τύπωσε επίσης τις νέες αποδοχές του. Τέλος με το τελευταίο `for loop` με τη βοήθεια της μεθόδου `getName()` επιστρέφετε το όνομα της κλάσης που δείχνει τον τύπο του κάθε εργαζομένου του πίνακα. Κάθε αντικείμενο στην Java γνωρίζει από ποια κλάση προέρχεται και μπορεί να πάρει πληροφορίες για αυτή μέσω της μεθόδου `getClass`.

Στον ορισμό των κλάσεων Employee, SalariedEmployee, HourlyEmployee, CommissionEmployee, BasePlusCommissionEmployee η πρώτη γραμμή περιέχει την εντολή `package payroll;` Αυτό σημαίνει ότι ανήκει στο πακέτο payroll. Δείτε από την εξερεύνηση τον φάκελο class του project TestPayroll. Θα έχει δημιουργηθεί ένας υποφάκελος payroll όπου εκεί αποθηκεύονται ta class files όλων των κλάσεων που περιέχουν την εντολή `package payroll;` Για να μπορέσει να βλέπει η εφαρμογή TestPayroll τις κλάσεις του πακέτου payroll θα πρέπει να εισάγω τις κλάσεις του πακέτου με την εντολή `import payroll.*;`

```
// Employee.java
// Employee abstract superclass.
package payroll;
public abstract class Employee {
   private String firstName;
   private String lastName;
   private String socialSecurityNumber;

   // constructor
   public Employee( String first, String last, String ssn )
   {
      firstName = first;
      lastName = last;
      socialSecurityNumber = ssn;
   }

   // set first name
   public void setFirstName( String first )
   {
      firstName = first;
   }

   // return first name
   public String getFirstName()
   {
      return firstName;
   }
```

```
   // set last name
   public void setLastName( String last )
   {
      lastName = last;
   }

   // return last name
   public String getLastName()
   {
      return lastName;
   }

   // set social security number
   public void setSocialSecurityNumber( String number )
   {
      socialSecurityNumber = number;  // should validate
   }

   // return social security number
   public String getSocialSecurityNumber()
   {
      return socialSecurityNumber;
   }

   // return String representation of Employee object
   public String toString()
   {
      return getFirstName() + " " + getLastName() +
         "\nsocial security number: " + getSocialSecurityNumber();
   }

   // abstract method overridden by subclasses
   public abstract double earnings();

} // end abstract class Employee
```

==========

```
// SalariedEmployee.java
// SalariedEmployee class extends Employee.
package payroll;
public class SalariedEmployee extends Employee {
   private double weeklySalary;

   // constructor
   public SalariedEmployee( String first, String last,
      String socialSecurityNumber, double salary )
   {
      super( first, last, socialSecurityNumber );
      setWeeklySalary( salary );
   }

   // set salaried employee's salary
   public void setWeeklySalary( double salary )
   {
      weeklySalary = salary < 0.0 ? 0.0 : salary;
   }

   // return salaried employee's salary
   public double getWeeklySalary()
   {
```

```
      return weeklySalary;
   }

   // calculate salaried employee's pay;
   // override abstract method earnings in Employee
   public double earnings()
   {
      return getWeeklySalary();
   }

   // return String representation of SalariedEmployee object
   public String toString()
   {
      return "\nsalaried employee: " + super.toString();
   }

} // end class SalariedEmployee


================================================================================

// HourlyEmployee.java
// HourlyEmployee class extends Employee.
package payroll;
public class HourlyEmployee extends Employee {
   private double wage;   // wage per hour
   private double hours;  // hours worked for week

   // constructor
   public HourlyEmployee( String first, String last,
      String socialSecurityNumber, double hourlyWage, double
hoursWorked )
   {
      super( first, last, socialSecurityNumber );
      setWage( hourlyWage );
      setHours( hoursWorked );
   }

   // set hourly employee's wage
   public void setWage( double wageAmount )
   {
      wage = wageAmount < 0.0 ? 0.0 : wageAmount;
   }

   // return wage
   public double getWage()
   {
      return wage;
   }

   // set hourly employee's hours worked
   public void setHours( double hoursWorked )
   {
      hours = ( hoursWorked >= 0.0 && hoursWorked <= 168.0 ) ?
         hoursWorked : 0.0;
   }

   // return hours worked
   public double getHours()
   {
      return hours;
   }
```

```
   // calculate hourly employee's pay;
   // override abstract method earnings in Employee
   public double earnings()
   {
      if ( hours <= 40 )  // no overtime
         return wage * hours;
      else
         return 40 * wage + ( hours - 40 ) * wage * 1.5;
   }

   // return String representation of HourlyEmployee object
   public String toString()
   {
      return "\nhourly employee: " + super.toString();
   }

} // end class HourlyEmployee
```

==========

```
// CommissionEmployee.java
// CommissionEmployee class extends Employee.
package payroll;
public class CommissionEmployee extends Employee {
   private double grossSales;      // gross weekly sales
   private double commissionRate;  // commission percentage

   // constructor
   public CommissionEmployee( String first, String last,
      String socialSecurityNumber,
      double grossWeeklySales, double percent )
   {
      super( first, last, socialSecurityNumber );
      setGrossSales( grossWeeklySales );
      setCommissionRate( percent );
   }

   // set commission employee's rate
   public void setCommissionRate( double rate )
   {
      commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
   }

   // return commission employee's rate
   public double getCommissionRate()
   {
      return commissionRate;
   }

   // set commission employee's weekly base salary
   public void setGrossSales( double sales )
   {
      grossSales = sales < 0.0 ? 0.0 : sales;
   }

   // return commission employee's gross sales amount
   public double getGrossSales()
   {
      return grossSales;
   }

   // calculate commission employee's pay;
   // override abstract method earnings in Employee
```

```
   public double earnings()
   {
      return getCommissionRate() * getGrossSales();
   }

   // return String representation of CommissionEmployee object
   public String toString()
   {
      return "\ncommission employee: " + super.toString();
   }

} // end class CommissionEmployee
```

==========

```
// BasePlusCommissionEmployee.java
// BasePlusCommissionEmployee class extends CommissionEmployee.
package payroll;
public class BasePlusCommissionEmployee extends CommissionEmployee {
   private double baseSalary;  // base salary per week

   // constructor
   public BasePlusCommissionEmployee( String first, String last,
      String socialSecurityNumber, double grossSalesAmount,
      double rate, double baseSalaryAmount )
   {
      super( first, last, socialSecurityNumber, grossSalesAmount,
rate );
      setBaseSalary( baseSalaryAmount );
   }

   // set base-salaried commission employee's base salary
   public void setBaseSalary( double salary )
   {
      baseSalary = salary < 0.0 ? 0.0 : salary;
   }

   // return base-salaried commission employee's base salary
   public double getBaseSalary()
   {
      return baseSalary;
   }

   // calculate base-salaried commission employee's earnings;
   // override method earnings in CommissionEmployee
   public double earnings()
   {
      return getBaseSalary() + super.earnings();
   }

   // return String representation of BasePlusCommissionEmployee
   public String toString()
   {
      return "\nbase-salaried commission employee: " +
         super.getFirstName() + " " + super.getLastName() +
         "\nsocial security number: " +
super.getSocialSecurityNumber();
   }

} // end class BasePlusCommissionEmployee
```

==========

```
// Payroll.java
// Employee hierarchy test program.
import java.text.DecimalFormat;
import javax.swing.JOptionPane;
import payroll.*;
public class TestPayroll {

   public static void main( String[] args )
   {
      DecimalFormat twoDigits = new DecimalFormat( "0.00" );

      // create Employee array
      Employee employees[] = new Employee[ 4 ];

      // initialize array with Employees
      employees[ 0 ] = new SalariedEmployee( "John", "Smith",
         "111-11-1111", 800.00 );
      employees[ 1 ] = new CommissionEmployee( "Sue", "Jones",
         "222-22-2222", 10000, .06 );
      employees[ 2 ] = new BasePlusCommissionEmployee( "Bob",
"Lewis",
         "333-33-3333", 5000, .04, 300 );
      employees[ 3 ] = new HourlyEmployee( "Karen", "Price",
         "444-44-4444", 16.75, 40 );

      String output = "";

      // generically process each element in array employees
      for ( int i = 0; i < employees.length; i++ ) {
         output += employees[ i ].toString();

         // determine whether element is a BasePlusCommissionEmployee
         if ( employees[ i ] instanceof BasePlusCommissionEmployee )
{

            // downcast Employee reference to
            // BasePlusCommissionEmployee reference
            BasePlusCommissionEmployee currentEmployee =
               ( BasePlusCommissionEmployee ) employees[ i ];

            double oldBaseSalary = currentEmployee.getBaseSalary();
            output += "\nold base salary: $" + oldBaseSalary;

            currentEmployee.setBaseSalary( 1.10 * oldBaseSalary );
            output += "\nnew base salary with 10% increase is: $" +
               currentEmployee.getBaseSalary();

         } // end if

         output += "\nearned $" + employees[ i ].earnings() + "\n";

      } // end for

      // get type name of each object in employees array
      for ( int j = 0; j < employees.length; j++ )
         output += "\nEmployee " + j + " is a " +
            employees[ j ].getClass().getName();

      JOptionPane.showMessageDialog( null, output );  // display
output
      System.exit( 0 );
      test t = new test();
```

```
        System.out.println("Hi");
      System.out.println(t.getClass().getName());

   } // end main

} // end class Payroll
```

Δυνατότητα Προσπέλασης	**public**	**protected**	**χωρίς προσδιοριστή**	**private**
Από την ίδια κλάση	✓	✓	✓	✓
Από τις κλάσεις του ίδιου πακέτου	✓	✓	✓	X
Από οποιαδήποτε κλάση εκτός του πακέτου όπου ανήκει η κλάση	✓	X	X	X
Από μια υποκλάση του ίδιου πακέτου	✓	✓	✓	X
Από μια υποκλάση έξω από το πακέτο όπου ανήκει η κλάση	✓	✓	X	X