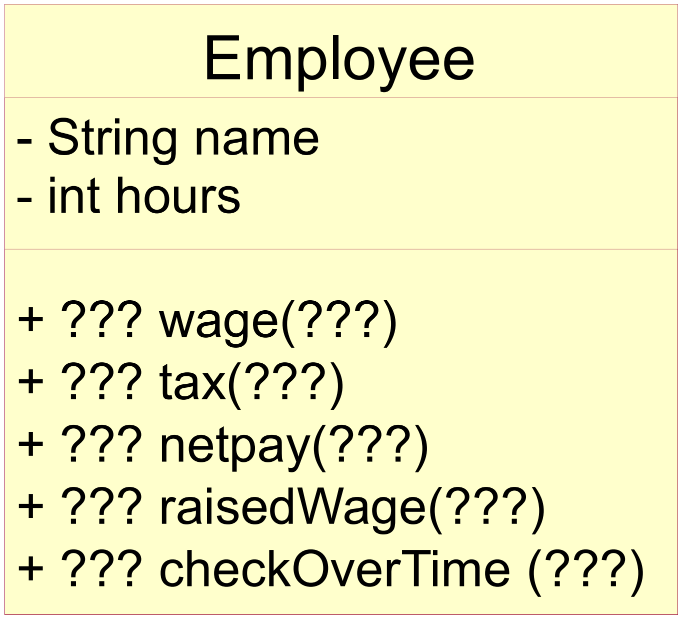
Excercises 2.1 – 2.7
Class Methods

Excersise 2.1

An employee has his name and the number of hours of work.

- Determines the wage (gross pay) of an employee from the number of hours of work. Suppose 12 dollars per hour.
- Utopia's tax accountants always use programs that compute income taxes even though the tax rate is a solid, never-changing 15%. Determine the tax on the gross pay.
- Also determine netpay of an employee from the number of hours worked base on gross pay and tax .
- Determine the raised wage to $14 for everyone
- No employee could possibly work more than 100 hours per week. To protect the company against fraud, the method should check that the hours doesn't exceed 100. If it does, the method returns false. Otherwise, it returns true

Class diagram



Employee

- String name
- int hours

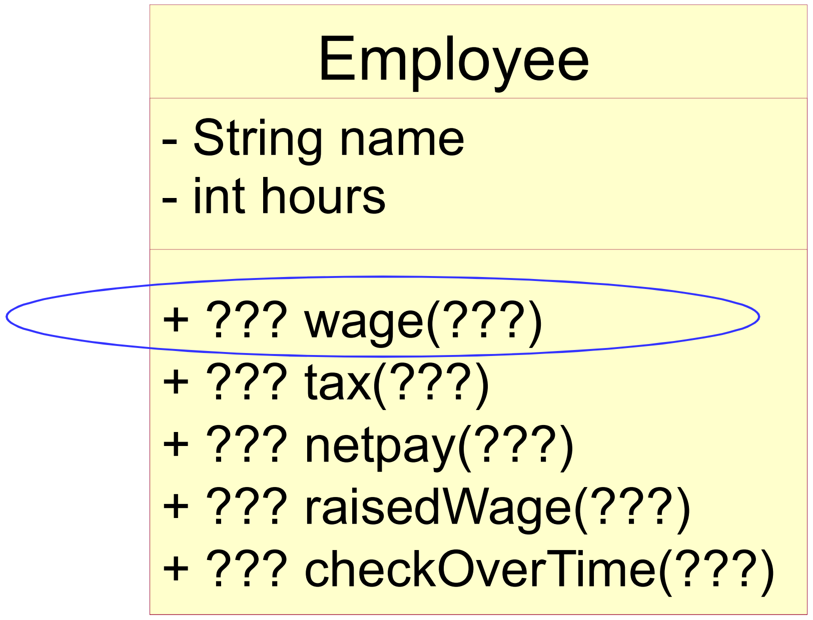
+ ??? wage(???)
+ ??? tax(???)
+ ??? netpay(???)
+ ??? raisedWage(???)
+ ??? checkOverTime (???)

Define Class, Contructor and Test

```
class Employee {
   String name;
   int hours;
   Employee(String name, int hours) {
      this.name = name;
      this.hours = hours;
   }
}
```

```
import junit.framework.*;
public class EmployeeTest extends TestCase {
   public void testContructor() {
      new Employee("Nam", 20);
      Employee aEmployee1 = new Employee("Mai", 30);
      Employee aEmployee2 = new Employee("Minh", 102);
   }
}
```

Compute `wage`



```
Employee
- String name
- int hours

+ ??? wage(???)
+ ??? tax(???)
+ ??? netpay(???)
+ ??? raisedWage(???)
+ ??? checkOverTime(???)
```

- Examples
  - Employee Nam works 20 hours and earns $240
  - Employee Mai works 30 hours and earns $360
  - Employee Minh works 102 hours and earns $1224

wage method template

```
class Employee {
   String name;
   int hours;
   Employee(String name, int     hours) {
     this.name = name;
     this.hours = hours;
   }

   // Determines the wage of an employee
   // from the number of hours of work
   int wage() {
      ...this.name...
      ...this.hours...
   }
}
```

wage method body

```
class Employee {
   String name;
   int hours;
   Employee(String name, int hours) {
      this.name = name;
      this.hours = hours;
   }

   // Determines the wage of an employee
   // from the number of hours of work
   int wage() {
      return this.hours * 12;
   }
}
```

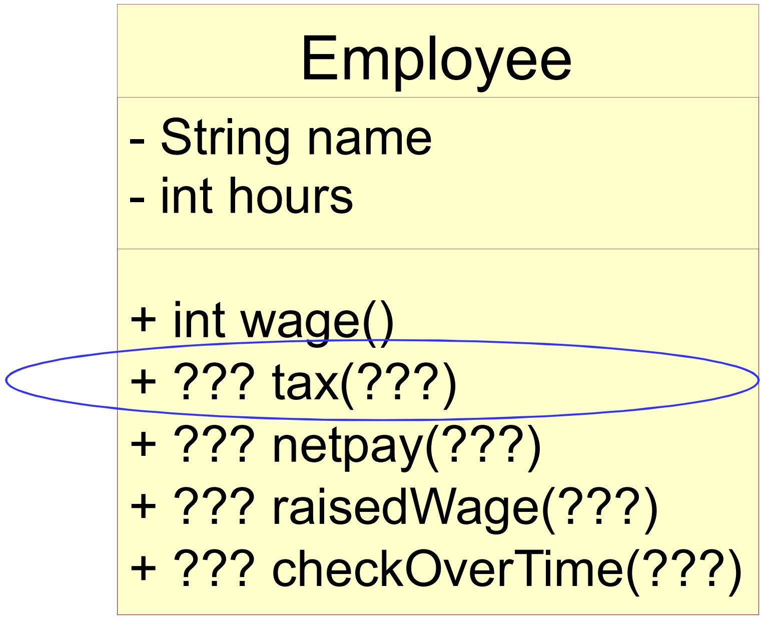
Test wage method

```
import junit.framework.*;

public class EmployeeTest extends TestCase {
   ...

   public void testWage() {
      Employee aEmployee1 = new Employee("Mai", 30);
      Employee aEmployee2 = new Employee("Minh", 102);
      assertEquals(aEmployee1.wage(), 360);
      assertEquals(aEmployee2.wage(), 1224);
      assertEquals(new Employee("Nam", 20).wage(), 240);
   }
}
```

Compute `tax`



- Examples
  - Employee Nam gets \$240 and has to pay \$36 for tax
  - Employee Mai gets \$360 and has to pay \$54 for tax
  - Employee Minh gets \$1224 and has to pay \$183.6 for tax

tax method template

```
class Employee {
   String name;
   int hours;
   Employee(String name, int hours) {
      this.name = name;
      this.hours = hours;
   }
   int wage() {
      return this.hours * 12;
   }

   // Determines the tax of an employee from the wage
   double tax() {
      ...this.name...
      ...this.hours...
      ...this.wage()...
   }
}
```

tax method implement

```java
class Employee {
   String name;
   int hours;
   Employee(String name, int hours) {
      this.name = name;
      this.hours = hours;
   }
   int wage() {
      return this.hours * 12;
   }

   // Determines the tax of an employee from the wage
   double tax() {
      // this.hours * 12 * 0.15;
      return this.wage() * 0.15;
   }
}
```

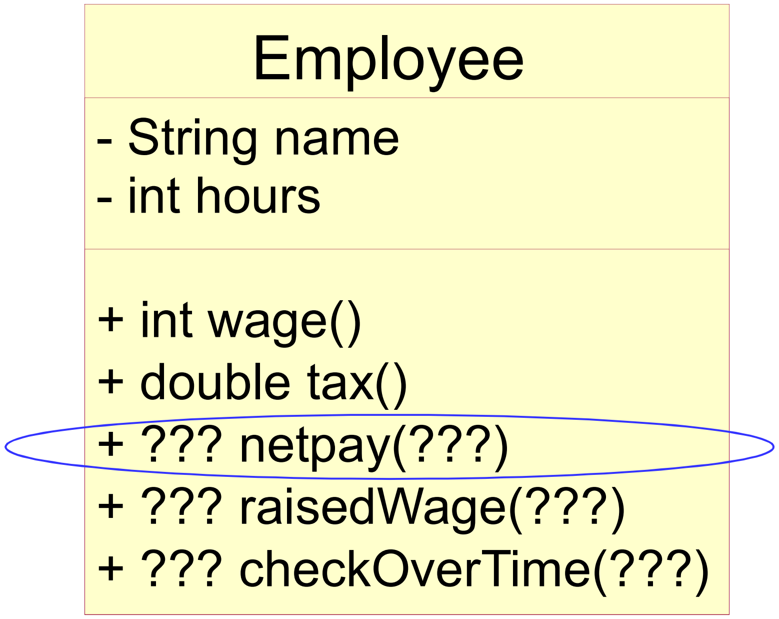
Test tax method

```
import junit.framework.*;

public class EmployeeTest extends TestCase {
   ...

   public void testTax() {
      Employee aEmployee1 = new Employee("Mai", 30);
      Employee aEmployee2 = new Employee("Minh", 102);
      assertEquals(aEmployee1.tax(), 54.0, 0.001);
      assertEquals(aEmployee2.tax(), 183.6, 0.001);
      assertEquals(new Employee("Nam", 20).tax(), 36.0, 0.001);
   }
}
```

Compute `netpay`



Employee

- String name
- int hours

+ int wage()
+ double tax()
+ ??? netpay(???)
+ ??? raisedWage(???)
+ ??? checkOverTime(???)

- Examples
  - With salary $240, Nam just receives $204 of netpay
  - With salary $360, Mai just receives $306 of netpay
  - With salary $1224, Minh just receives $1020 of netpay

netpay method template

```
class Employee {
   String name;
   int hours;
   Employee(String name, int     hours) {
      this.name = name;
      this.hours = hours;
   }

   // Determines the netpay of an employee
   double netpay() {
      ...this.name...this.hours...
      ...this.wage()...
      ...this.tax()...
   }
}
```

netpay method implement

```java
class Employee {
   String name;
   int hours;
   Employee(String name, int hours) {
      this.name = name;
      this.hours = hours;
   }
   int wage() {
      return this.hours * 12;
   }
   double tax() {
      return this.wage()) * 0.15;
   }
   // Determines the netpay of an employee
   double netpay() {
      return this.wage() - this.tax();
   }
}
```

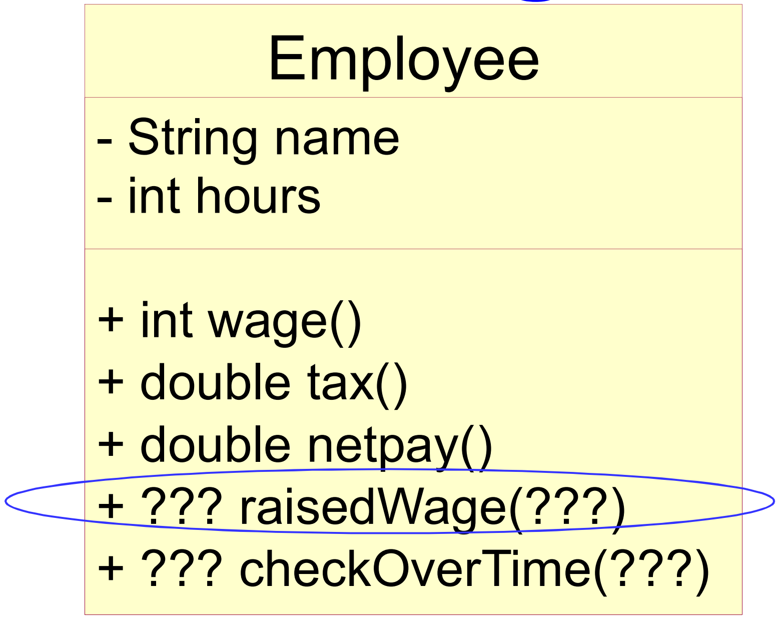
Test netpay method

```
import junit.framework.*;

public class TestEmployee extends TestCase {
   ...

   public void testNetpay() {
      Employee aEmployee1 = new Employee("Mai", 30);
      Employee aEmployee2 = new Employee("Minh", 102);
      assertEquals(aEmployee1.netpay(), 306.0, 0.01);
      assertEquals(aEmployee2.netpay(), 1040.4, 0.01);
      assertEquals(new Employee("Nam", 20).netpay(),
                   204.0, 0.001);
   }
}
```

Compute `raisedWage`



- Examples
  - With basic salary $240, after getting bonus, total income of Nam is $254
  - With basic salary $360, after getting bonus, total salary of Mai is $374
  - With basic salary $1224, after getting bonus, total salary of Minh is $1238

raisedWage method template

```
class Employee {
   String name;
   int hours;
   Employee(String name, int     hours) {
      this.name = name;
      this.hours = hours;
   }

   // Raise the wage of employee with 14$
   double raisedWage() {
      ...this.name...this.hours...
      ...this.wage()...this.tax()...
      ...this.netpay()...
   }
}
```

raisedWage method implement

```
class Employee {
   String name;
   int hours;
   ...
   int wage() {
      return this.hours * 12;
   }
   double tax() {
      return this.wage() * 0.15;
   }
   double netpay() {
      return this.wage() - this.tax();
   }
   // Raise the wage of employee with 14 $
   double raisedWage() {
      return this.wage() + 14;
   }
}
```

Test raisedWage method

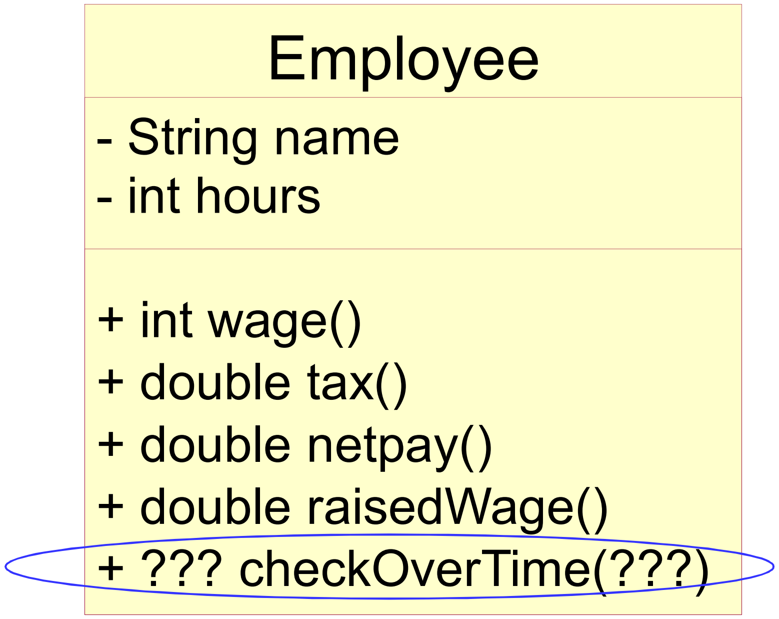
```
import junit.framework.*;

public class TestEmployee extends TestCase {
  ...

  public void testRaisedWage() {
    Employee aEmployee1 = new Employee("Mai", 30);
    Employee aEmployee2 = new Employee("Minh", 102);
    assertEquals(aEmployee1.raisedWage(), 374.0, 0.001);
    assertEquals(aEmployee2.raisedWage(), 1238.0, 0.001);
    assertEquals(new Employee("Nam", 20).raisedWage(),
                 254, 0.001);

  }
}
```

Compute **wage**



- Examples
  - It is false that Nam and Mai work 20 and 30 hours per week
  - It is true for Minh to work 102 hours per week

checkOverTime method template

```
class Employee {
   String name;
   int hours;
   Employee(String name, int     hours) {
      this.name = name;
      this.hours = hours;
   }

   // Determines whether the number of hours of work
   // exceeds 100
   boolean checkOverTime() {
      ...this.name...this.hours...
      ...this.wage()...this.tax()...
      ...this.netpay()...this.raisedWage()
   }
}
```

checkOverTime method implement

```java
class Employee {
   String name;
   int hours;
   Employee(String name, int     hours) {
      this.name = name;
      this.hours = hours;
   }
   ...

   // Determines whether the number of hours of work
   // exceeds 100
   boolean checkOverTime() {
      return this.hours > 100;
   }
}
```

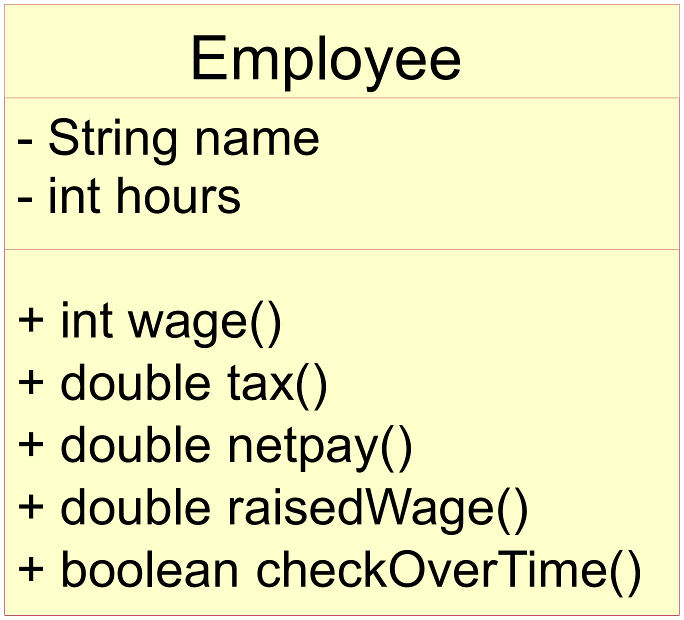
Test checkOverTime method

```java
import junit.framework.*;

public class EmployeeTest extends TestCase {
   ...

   public void testCheckOverTime() {
      Employee aEmployee1 = new Employee("Mai", 30);
      Employee aEmployee2 = new Employee("Minh", 102);
      assertFalse(aEmployee1.checkOverTime());
      assertTrue(aEmployee2.checkOverTime());
      assertFalse(new Employee("Nam", 20).checkOverTime());
   }
}
```

Class diagram - Final



Exercise 2.1.1 (extended)

Adding to **Employee** exercise:

- Develop the method tax, which consumes the gross pay and produces the amount of tax owed.
  For a gross pay of $240 or less, the tax is 0%;
  for over $240 and $480 or less, the tax rate is 15%;
  and for any pay over $480, the tax rate is 28%.
- Also develop netpay.
  The method determines the net pay of an employee from the number of hours worked. The net pay is the gross pay minus the tax. Assume the hourly pay rate is $12

Solution 2.1.1: `taxWithRate()`

```
double taxWithRate() {
   double grossPay = this.wage();
   if (grossPay <= 240)
     return 0.0;
   if (grossPay <= 480)
     return grossPay * 0.15;
   return grossPay * 0.28;
}
```

```
public void testTaxWithRate() {
   assertEquals(new Employee("Nam", 20).taxWithRate(),
                     0.0, 0.001);
   Employee aEmployee1 = new Employee("Mai", 30);
   assertEquals(aEmployee1.taxWithRate(), 54.0, 0.001);
   Employee aEmployee2 = new Employee("Minh", 102);
   assertEquals(aEmployee2.taxWithRate(), 342.72, 0.001);
}
```

Solution 2.1.1: `netpayWithRate()`

Method implementation

```
double netpayWithRate() {
    return this.wage() - this.taxWithRate();
}
```

Unit testing

```
public void testNetpayWithRate() {
    assertEquals(new Employee("Nam", 20).netpayWithRate(),
                 240.0, 0.001);
    Employee aEmployee1 = new Employee("Mai", 30);
    assertEquals(aEmployee1.netpayWithRate(), 306.0, 0.001);
    Employee aEmployee2 = new Employee("Minh", 102);
    assertEquals(aEmployee2.netpayWithRate(), 881.28, 0.001);
}
```

Exercise 2.2

- An old-style movie theater has a simple profit method. Each customer pays for ticket, for example $5. Every performance costs the theater some money, for example $20 and plus service charge per attendee, for example $.50.
- Develop the totalProfit method. It consumes the number of attendees (of a show) and produces how much income the attendees profit
- Example:
  - **totalProfit(40)** return **$160**

Solution

Exercise 2.3

- Take a look at this following class:

```java
// represent information about an image
class Image {
   int width; // in pixels
   int height; // in pixels
   String source; // file name
   String quality; // informal
   Image(int width, int height,
             String source, String quality) {
      this.width = width;
      this.height = height;
      this.source = source;
      this.quality = quality;
   }

}
```

Exercise 2.3 (cont): Design methods

- `isPortrait`, which determines whether the image's height is larger than its width;
- `size`, which computes how many pixels the image contains;
- `isLarger`, which determines whether one image contains more pixels than some other image; and
- `same`, which determines whether this image is the same as a given one.
- `sizeString` produces one of three strings, depending on the number of pixels in the image:
  - "small" for images with 10,000 pixels or fewer;
  - "medium" for images with between 10,001 and 1,000,000 pixels;
  - "large" for images that are even larger than that.

Exercise 2.4

Modify the *Coffee* class so that ***cost*** takes into account bulk discounts:

. . . Develop a program that computes the cost of selling bulkcoffee at a specialty coffee seller from a receipt that includes the kind of coffee, the unit price, and the total amount (weight) sold. If the sale is for less than 5,000 pounds, there is no discount. For sales of 5,000 pounds to 20,000 pounds, the seller grants a discount of 10%. For sales of 20,000 pounds or more, the discount is 25%. . . .

Exercise 2.5

- Design the class ***JetFuel***, whose purpose it is to represent the sale of some quantity of jet fuel.
- Each instance contains the **quantity sold** (in integer gallons), the **quality level** (a string), and the current **base price** of jet fuel (in integer cents per gallon). The class should come with two methods:
  - ***totalCost***, which computes the cost of the sale,
  - ***discountPrice***, which computes the discounted price. The buyer gets a 10% discount if the sale is for more than 100,000 gallons

Exercise 2.6

- Given a quadratic equation with coefficients a, b, and c.
- Develop **whatKind** method.
  It then determines whether the equation is degenerate and, if not, how many solutions the equation has. The method produces one of four symbols: "degenerate", "two", "one", or "none".

Solution

Exercise 2.7

Information about the transaction in bank includes customer **name**, and **deposit** amount and **maturity** (computed in year)

2.7.1 Develop the method **interest**. It consumes a deposit amount and produces the actual amount of interest that the money earns in a year. The bank pays a flat 4% per year for deposits of up to $1,000, a flat 4.5% for deposits of up to $5,000, and a flat 5% for deposits of more than $5,000

Solution

Exercise 2.7 (cont)

Some credit card companies pay back a small portion of the charges a customer makes over a year. One company returns

- 0.25% for the first $500 of charges,
- 0.50% for the next $1000 (that is, the portion between $500 and $1500),
- 0.75% for the next $1000 (that is, the portion between $1500 and $2500), and 1.0% for everything above $2500.

2.7.2 Define the **payback** method, which consumes a charge amount and computes the corresponding pay-back amount.

Solution

Relax...

& Do Exercise

Solution 2.2

```
class MovieShow {
   double ticketPrice;
   double performanceCost;
   double chargePerAttendee;
   MovieShow(double ticketPrice, double performanceCost,
             double chargePerAttendee) {
      this.ticketPrice = ticketPrice;
      this.performanceCost = performanceCost;
      this.chargePerAttendee = chargePerAttendee;
   }
   double cost(int numAttendee) {
      return this.performanceCost + this.chargePerAttendee * numAttendee;
   }
   double revenue(int numAttendee) {
      return this.ticketPrice * numAttendee;
   }
   double totalProfit(int numAttendee) {
      return this.revenue(numAttendee)- this.cost(numAttendee);
   }
}
```

Solution 2.2 (cont): Using test

```java
public void testTotalProfit() {
   MovieShow aMovie1 = new MovieShow(5.0, 20.0, 0.15);
   MovieShow aMovie2 = new MovieShow(6.0, 40.0, 0.1);
   MovieShow aMovie3 = new MovieShow(7.0, 50.0, 0.2);

   assertEquals(465.0, aMovie1.totalProfit(100), 0.001);
   assertEquals(550.0, aMovie2.totalProfit(100), 0.001);
   assertEquals(630.0, aMovie3.totalProfit(100), 0.001);
}
```

Back

Solution 2.6: Class definition

```
class Quadratic {
   double a;
   double b;
   double c;
   Quadratic(double a, double b, double c) {
      this.a = a;
      this.b = b;
      this.c =c;
   }
   private computeDelta() {
      return this.b * this.b - 4 * this.a * this.c;
   }
   String whatKind() {
      double delta = this.computeDelta();
      if (this.a == 0) return "degenerate";
      if (delta < 0) return "none";
      if (delta == 0) return "one solution";
      return "two solution";
   }
}
```

Solution 2.6 (cont): Using test

```java
public void testWhatKind() {
   Quadratic q1= new Quadratic(0.0, 1.0, 2.0);
   Quadratic q2= new Quadratic(2.0, 1.0, 2.0);
   Quadratic q3= new Quadratic(1.0, 2.0, 1.0);
   Quadratic q4= new Quadratic(2.0, 3.0, 1.0);

   assertEquals("degenerate", q1.whatKind());
   assertEquals("none", q2.whatKind());
   assertEquals("one solution", q3.whatKind());
   assertEquals("two solution", q4.whatKind());
}
```

Back

Solution 2.7.1: Class definition

```java
class Transaction {
   String customerName;
   double depositeAmount;
   int maturity;
   Transaction(String customerName, double depositeAmount,
               int maturity) {
      this.customerName = customerName;
      this.depositeAmount = depositeAmount;
      this. maturity = maturity;
   }

   double interest() {
      if (this.depositeAmount <= 1000)
         return this.depositeAmount * 0.04;
      if (this.depositeAmount <= 5000)
         return this.depositeAmount * 0.045 ;
      return this.depositeAmount * 0.05 ;
   }
}
```

Solution 2.7.1 (cont): Using test

```
public void testInterest(){
   Transaction t1 = new Transaction("Thuy", 6000, 2);
   Transaction t2 = new Transaction("Mai", 2500, 1);
   Transaction t3 = new Transaction("Nam", 1500, 2);
   Transaction t4 = new Transaction("Tien", 500, 2);

   assertEquals(300.0, t1.interest(), 0.001);
   assertEquals(112.5, t2.interest(), 0.001);
   assertEquals(67.5, t3.interest(), 0.001);
   assertEquals(20.0, t4.interest(), 0.001);
}
```

Back

Solution 2.7.2: Method implementation

```
double payback() {
   if (this.depositeAmount <= 500)
      return this.depositeAmount * 0.0025;
   if (this.depositeAmount <= 1500)
      return 500 * 0.0025 + (this.depositeAmount - 500)* 0.005 ;
   if (this.depositeAmount <= 2500 )
      return 500 * 0.0025 + 1000 * 0.005 +
          (this.depositeAmount -1500)* 0.0075;
   return 500 * 0.0025 + 1000 * 0.005 + 1000 * 0.0075 +
          (this.depositeAmount - 2500)* 0.01;

}
```

Solution 2.7.2 (cont) Using test

```
public void testPayback() {
   Transaction t1 = new Transaction("Thuy", 6000, 2);
   Transaction t2 = new Transaction("Mai", 2500, 1);
   Transaction t3 = new Transaction("Nam", 1500, 2);
   Transaction t4 = new Transaction("Tien", 500, 2);

   assertEquals(48.75, t1.payback(), 0.001);
   assertEquals(13.75, t2.payback(), 0.001);
   assertEquals(6.25, t3.payback(), 0.001);
   assertEquals(1.25, t4.payback(), 0.001);
}
```

Back