King Saud University
College of Computer and Information Sciences
Department of Computer Science
CSC113 – Computer Programming II – Final Exam – Fall 2015

Exercise 1:

What would be the output of the following Java program.

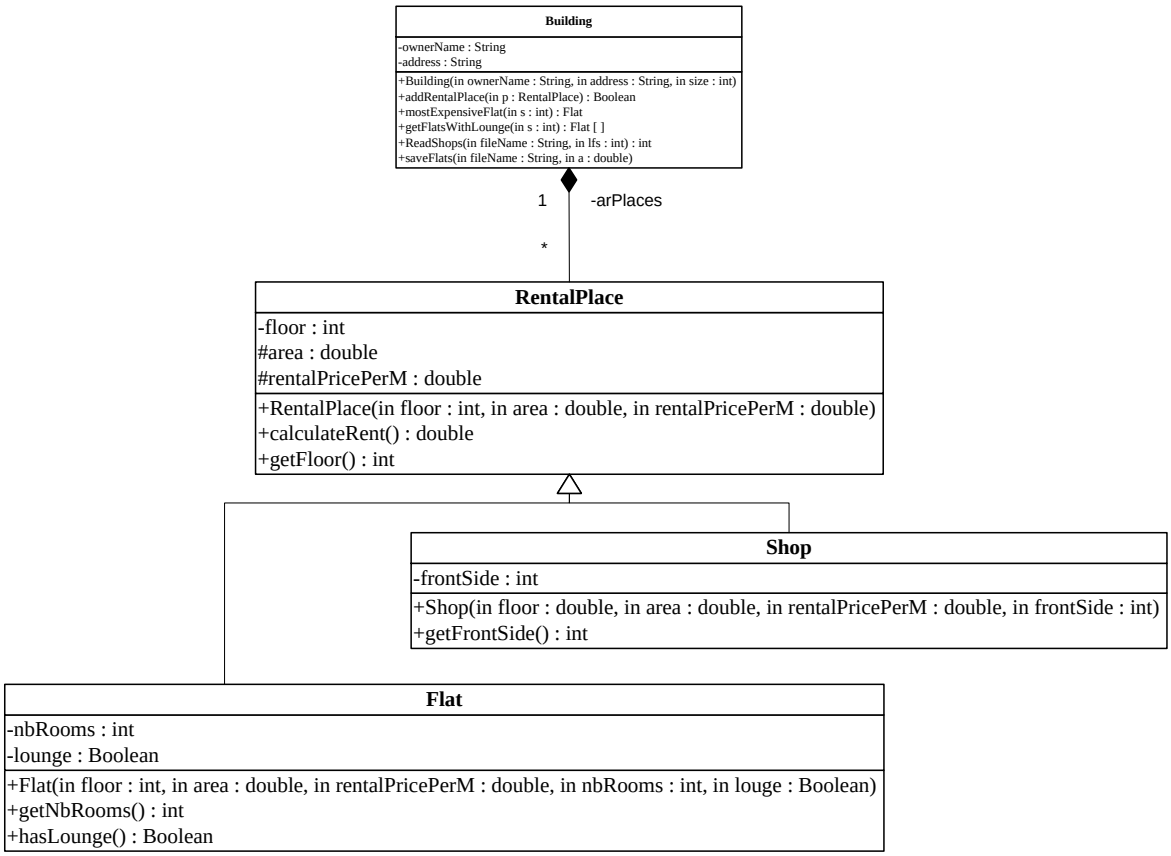
```
abstract class Shape {
      protected String name;

      public Shape() { name="Shape";
                       System.out.println("Shape Object Created");
      }
      public Shape( String n) { name = n;
                       System.out.println(name +" Object Created ");
      }
      public abstract void area( );
}

class Rectangle extends Shape {
            private double length;
            private double width;

            public Rectangle(){ length=0;    width=0;
                       System.out.println("Rectangle Object Created ");
            }
            public Rectangle(String n,double len, double w)
                    throws IllegalArgumentException {
                  super(n);

                  if (len< 0 || w<0)
                        throw new IllegalArgumentException (
                                    "Length or width cannot be zero");

                  length=len;      width=w;
                  System.out.println
                  ("Rectangle Created: Length="+len+"and  width= "+w);
            }

            public void area( ) throws ArithmeticException   {
                  double area=length*width;
                  System.out.println("Area is = " + area);

                  if(area > 80)
                        throw new ArithmeticException("Too big Rect.");
            }
      }
```

```
public class Main {
 public static void main(String[] args){
      double len[] = {12.0, 12.0, 4.0, 3.0};
      double wid[] = {4.0, 12.0, -4.0, 3.0}
      Shape shape[] = new Shape[4];
      shape[0]=new Rectangle();

      for(int i=1; i<5; i++)  {
            System.out.println("Iteration " + (i));
            try  {
                  shape[i]=new Rectangle("Rec",len[i-1], wid[i-1]);
                  shape[i].area();
           }
            catch(ArrayIndexOutOfBoundsException e){
                  System.out.println("Wrong index "+i); }
            catch(IllegalArgumentException e){
                  System.out.println(e.getMessage()); }
            catch(Exception e) {
                  System.out.println(e.getMessage()); }
      }
  }
 }
```

Solution

```
Shape Object Created
Rectangle Object Created
Iteration 1
Rec Object Created
Rectangle Created: Length=12.0and  width= 4.0
Area is = 48.0
Iteration 2
Rec Object Created
Rectangle Created: Length=12.0and  width= 12.0
Area is = 144.0
Too big Rect.
Iteration 3
Rec Object Created
Length or width cannot be zero
Iteration 4
Rec Object Created
Rectangle Created: Length=3.0and  width= 3.0
Wrong index 4
```

Exercise2:



RentalPlace class***:***

- Attributes:
  - ***floor***: the floor number of the rental place.
  - ***area:*** the area of the rental place.
  - ***rentalPricePerM:*** the rental price per meter square of the rental place.
- Methods:
  - ***rentalPlace (floor:*** *int****, area:*** *double****, rentalPricePerM:*** *double****)***: constructor
  - ***calculateRent()*:** this method returns the annual rent of the rental place. This rent is computed as following:
    - ***For Flat:*** the rent = area * rentalPricePerM + nbRooms * 1000.
    - ***For Shop:*** the rent = area * rentalPricePerM + frontSide * 3000.

Flat class

- Attributes:
  - ***nbRooms:*** the number of rooms in the flat.
  - ***lounge***: It is a Boolean that indicates whether the flat has a lounge or not.

- Methods:
  - ***Flat (floor:*** *int****, area:*** *double****, rentalPricePerM:*** *double****, nbRooms:*** *int****, lounge:*** *boolean* ***)***: constructor.
  - ***getNbRooms***: this method returns the number of rooms of the flat.
  - ***hasLounge***: this method returns true if the flat has a lounge, false otherwise.

Shop class

- Attributes:
  - ***frontSide:*** the front side of the shop in meter.
- Methods:
  - ***Shop (floor:*** *int****, area:*** *double****, rentalPricePerM:*** *double****, frontSide:*** *int****)***: constructor.
  - ***getFrontSide()***: this method returns the front side of the shop.

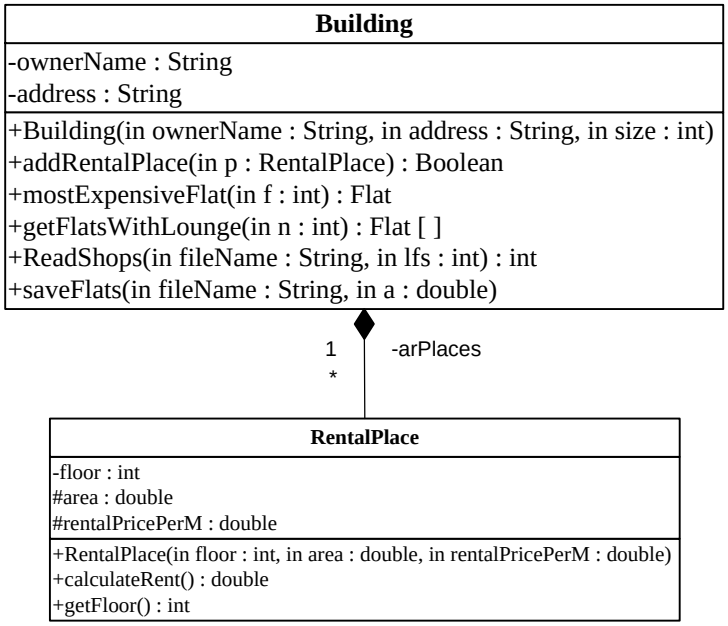
QUESTION: Translate into Java code the class ***RentalPlace*** and the class ***Shop***.

Solution

```
public abstract class RentalPlace {
	private int floor;
	protected double area;
	protected double rentalPricePerM;

	public RentalPlace(int f, double a, double r) {
		floor = f;		area = a;
		rentalPricePerM = r;
	}

	public abstract double calculateRent();

	public int getFloor() { return floor; }
}

public class Shop extends RentalPlace {
	private int frontSide;

	public Shop (int f, double a, double r, int fs) {
		super(f, a, r);
		frontSide  = fs;
	}

	public double calculateRent() {
		return ((area * rentalPricePerM) + (frontSide * 3000.0) );
	}
	public int getFrontSide() {	return frontSide; }

}
```

Exercise 3:

Let's consider the same class ***RentalPlace*** and its sub-classes as described in exercise 2.



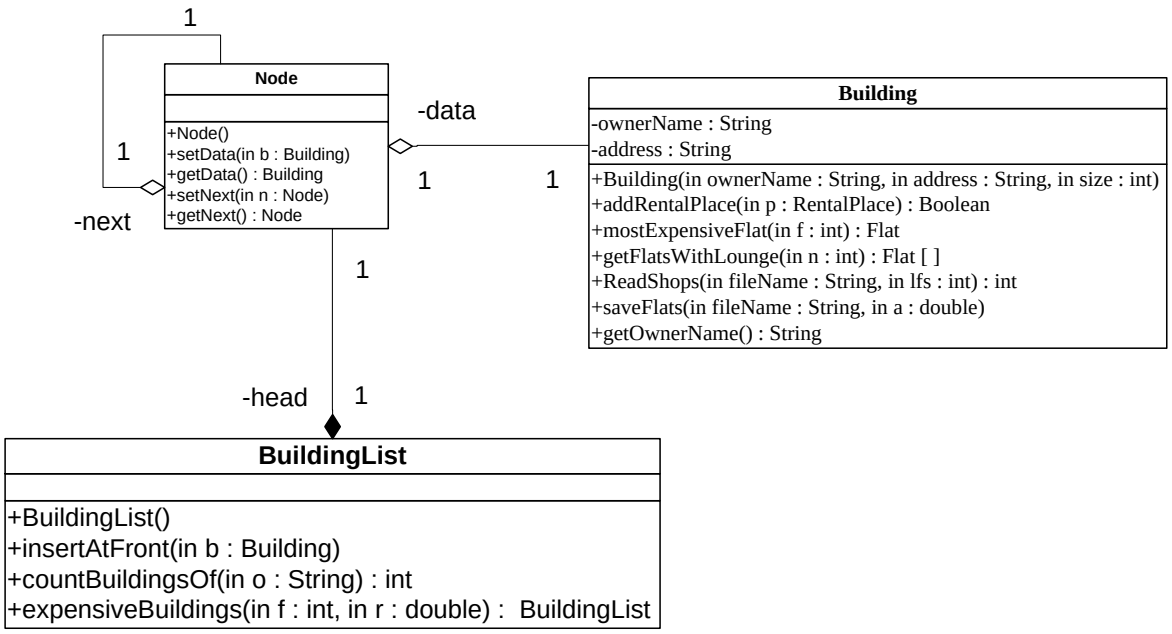
Building class***:***

- Attributes:
  - ***ownerName***: the name of the Building.
  - ***address***: the address of the Building.
- Methods:
  - ***Building(ownerName:*** *String****, address:*** *String****, size:*** *int****)***: constructor
  - ***addRentalPlace (p:*** *RentalPlace****):*** this method adds the rental place ***p*** to the Building. This method raises an ***`IllegalArgumentException`*** if the shop is not in the ground floor (Notice that the ground floor number is zero). This method raises an ***`ArrayIndexOutOfBoundsException`*** when it is not possible to add ***p*** to the building.
  - ***mostExpensiveFlat (f:*** *int****):*** this method returns the most expensive flat in the floor ***f***.
  - ***getFlatsWithLounge(n:*** *int****):*** this method returns an array containing all flats with lounge having a number of rooms greater than ***n***.
  - ***readShops(fileName: string, lfs: int):*** this method reads shop objects having the front side greater than ***lfs*** from the file ***fileName*** and adds them to the building. It returns the number of added shops.
  - ***saveFlats(fileName: string , a: double):*** this method saves all flat objects having an area equal to ***a*** in the file ***fileName***.

QUESTION: Translate into Java code the class ***Building.***

Exercise 4:

Let's consider the same class ***Building*** described in exercise 3.



BuildingList class***:***

- Methods:
  - ***BuildingList ()***: constructor
  - ***insertAtFront (b: Building*)*:** this method adds the Building ***b*** at the beginning of the list.
  - ***countBuildingsOf (o: String*)*:** this method counts and returns the number of buildings that belong to the owner ***o***.
  - ***expensiveBuildings(f:*** int, ***r:*** double): This method returns a linked list containing all buildings where the rent of the most expensive flat in the floor ***f*** is greater than ***r***.

QUESTION: Translate into Java code the class ***BuildingList.***

King Saud University
College of Computer and Information Sciences
Department of Computer Science
CSC113 – Computer Programming II – Final Exam – Fall 2015

Exercise 1:

What would be the output of the following Java program.

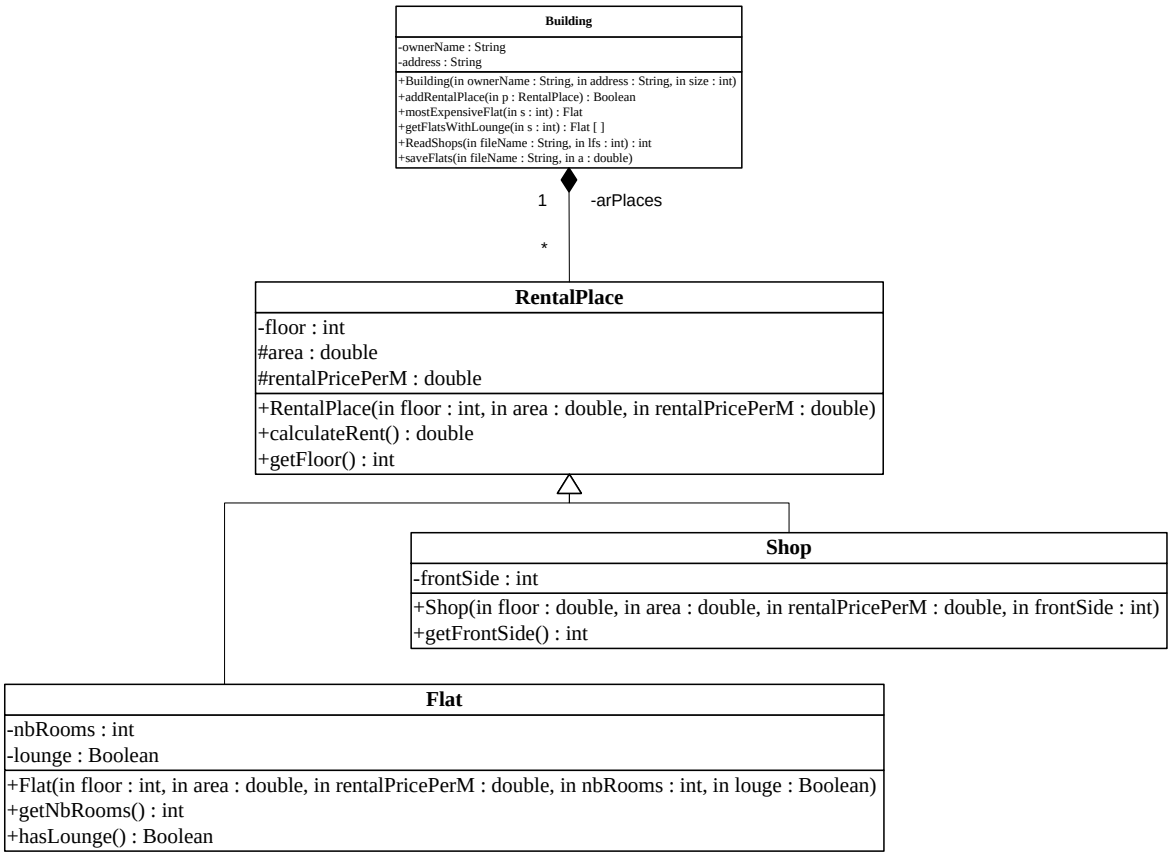
```
abstract class Shape {
      protected String name;

      public Shape() { name="Shape";
                       System.out.println("Shape Object Created");
      }
      public Shape( String n) { name = n;
                       System.out.println(name +" Object Created ");
      }
      public abstract void area( );
}

class Rectangle extends Shape {
            private double length;
            private double width;

            public Rectangle(){ length=0;    width=0;
                       System.out.println("Rectangle Object Created ");
            }
            public Rectangle(String n,double len, double w)
                    throws IllegalArgumentException {
                  super(n);

                  if (len< 0 || w<0)
                        throw new IllegalArgumentException (
                                    "Length or width cannot be zero");

                  length=len;      width=w;
                  System.out.println
                  ("Rectangle Created: Length="+len+"and  width= "+w);
            }

            public void area( ) throws ArithmeticException   {
                  double area=length*width;
                  System.out.println("Area is = " + area);

                  if(area > 80)
                        throw new ArithmeticException("Too big Rect.");
            }
      }
```

```
public class Main {
 public static void main(String[] args){
      double len[] = {12.0, 12.0, 4.0, 3.0};
      double wid[] = {4.0, 12.0, -4.0, 3.0}
      Shape shape[] = new Shape[4];
      shape[0]=new Rectangle();

      for(int i=1; i<5; i++)  {
            System.out.println("Iteration " + (i));
            try  {
                  shape[i]=new Rectangle("Rec",len[i-1], wid[i-1]);
                  shape[i].area();
           }
            catch(ArrayIndexOutOfBoundsException e){
                  System.out.println("Wrong index "+i); }
            catch(IllegalArgumentException e){
                  System.out.println(e.getMessage()); }
            catch(Exception e) {
                  System.out.println(e.getMessage()); }
      }
  }
 }
```

Solution Ex1: -18*0.25---------- /5

```
Shape Object Created
Rectangle Object Created
Iteration 1
Rec Object Created
Rectangle Created: Length=12.0and  width= 4.0
Area is = 48.0
Iteration 2
Rec Object Created
Rectangle Created: Length=12.0and  width= 12.0
Area is = 144.0
Too big Rect.
Iteration 3
Rec Object Created
Length or width cannot be zero
Iteration 4
Rec Object Created
Rectangle Created: Length=3.0and  width= 3.0
Wrong index 4
```

Exercise2:



RentalPlace class***:***

- Attributes:
  - ***floor***: the floor number of the rental place.
  - ***area:*** the area of the rental place.
  - ***rentalPricePerM:*** the rental price per meter square of the rental place.
- Methods:
  - ***rentalPlace (floor:*** *int****, area:*** *double****, rentalPricePerM:*** *double****)***: constructor
  - ***calculateRent()*:** this method returns the annual rent of the rental place. This rent is computed as following:
    - ***For Flat:*** the rent = area * rentalPricePerM + nbRooms * 1000.
    - ***For Shop:*** the rent = area * rentalPricePerM + frontSide * 3000.

Flat class

- Attributes:
  - ***nbRooms:*** the number of rooms in the flat.
  - ***lounge***: It is a Boolean that indicates whether the flat has a lounge or not.

- Methods:
  - ***Flat (floor:*** *int****, area:*** *double****, rentalPricePerM:*** *double****, nbRooms:*** *int****, lounge:*** *boolean* ***)***: constructor.
  - ***getNbRooms***: this method returns the number of rooms of the flat.
  - ***hasLounge***: this method returns true if the flat has a lounge, false otherwise.

Shop class

- Attributes:
  - ***frontSide:*** the front side of the shop in meter.
- Methods:
  - ***Shop (floor:*** *int****, area:*** *double****, rentalPricePerM:*** *double****, frontSide:*** *int****)***: constructor.
  - ***getFrontSide()***: this method returns the front side of the shop.

QUESTION: Translate into Java code the class ***RentalPlace*** and the class ***Shop***.

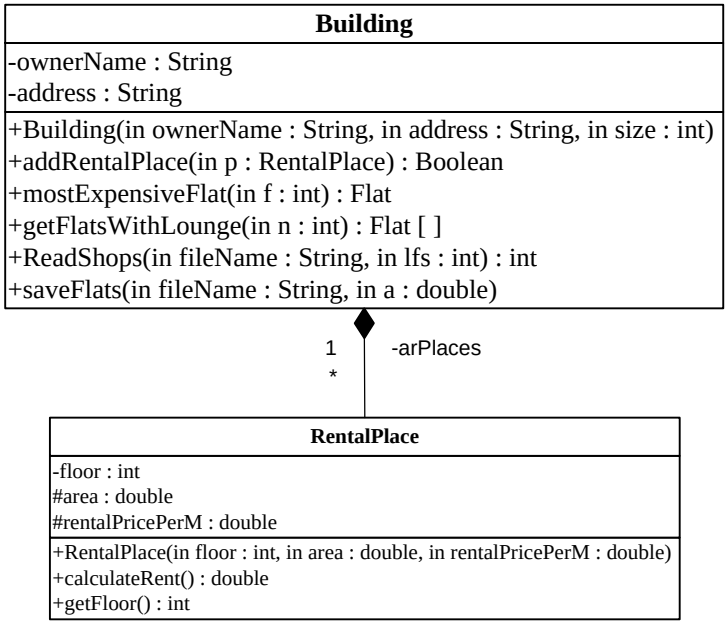
Solution Ex2: ---------- /10

```
public abstract class RentalPlace { ---------- 1 ---------- /4
      private int floor;
      protected double area;
      protected double rentalPricePerM;

      public RentalPlace(int f, double a, double r) {
            floor = f;        area = a;
            rentalPricePerM = r;
      }

      public RentalPlace(RentalPlace p) {  ---------- 1
            floor = p.floor;        area = p.area;
            rentalPricePerM = p.rentalPricePerM;
      }

      public abstract double calculateRent();  ---------- 1

      public int getFloor() { return floor;     }  ---------- 1

      public double getArea() { return area; }
}
```

```
public class Shop extends RentalPlace {  ---------- 1  ---------- /6
	private int frontSide;

	public Shop (int f, double a, double r, int fs) {
		super(f, a, r);  ---------- 1
		frontSide  = fs;
	}

	public Shop (Shop s) {  ---------- 1
		super(s); ---------- 1
		frontSide  = s.frontSide;
	}

	public double calculateRent() {  ---------- 1
		return ((area * rentalPricePerM) + (frontSide * 3000.0) );
	}
	public int getFrontSide() {  ---------- 1
		return frontSide;
	}
}
```

Exercise 3:

Let's consider the same class ***RentalPlace*** and its sub-classes as described in exercise 2.



Building class***:***

- Attributes:
  - ***ownerName***: the name of the Building.
  - ***address***: the address of the Building.
- Methods:
  - ***Building(ownerName:*** *String****, address:*** *String****, size:*** *int****)***: constructor
  - ***addRentalPlace (p:*** *RentalPlace****):*** this method adds the rental place ***p*** to the Building. This method raises an ***`IllegalArgumentException`*** if the shop is not in the ground floor (Notice that the ground floor number is zero). This method raises an ***`ArrayIndexOutOfBoundsException`*** when it is not possible to add ***p*** to the building.
  - ***mostExpensiveFlat (f:*** *int****):*** this method returns the most expensive flat in the floor ***f***.
  - ***getFlatsWithLounge(n:*** *int****):*** this method returns an array containing all flats with lounge having a number of rooms greater than ***n***.
  - ***readShops(fileName: string, lfs: int):*** this method reads shop objects having the front side greater than ***lfs*** from the file ***fileName*** and adds them to the building. It returns the number of added shops.
  - ***saveFlats(fileName: string , a: double):*** this method saves all flat objects having an area equal to ***a*** in the file ***fileName***.

QUESTION: Translate into Java code the class ***Building.***

Solution

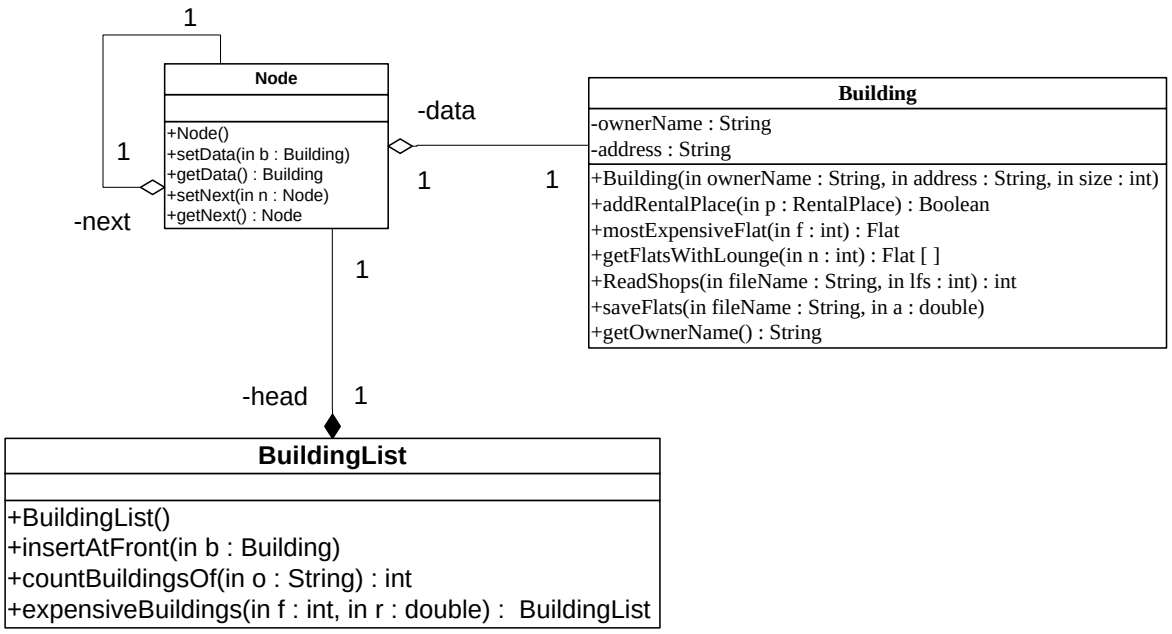
```
public class Building {  ---------- /45
      private String ownerName;
      private String address;

      private RentalPlace[] arPlaces;  ---------- 1
      private int nbPlaces;  ---------- 1

      public Building(String o, String a, int size) { ---------- /2
            ownerName = o;          address = a;
            arPlaces = new RentalPlace[size];   ---------- 1
            nbPlaces = 0;  ---------- 1
      }

      public boolean addRentalPlace(RentalPlace p) throws  ---------- 1
                                          IllegalArgumentException, ---- /9
                                          ArrayIndexOutOfBoundsException {

            if (nbPlaces < arPlaces.length) {  ---------- 1
                  if (p instanceof Shop) {  ---------- 1
                        if (p.getFloor()!= 0)throw ---------- 1
                              new IllegalArgumentException("Error");
                        else
                              arPlaces[nbPlaces] = new Shop((Shop) p); ---1+1
                  }
                  else
                        arPlaces[nbPlaces] = new Flat( (Flat) p); ------ 1+1

                  nbPlaces++;  ---------- 1
            }
            else
                  throw new ArrayIndexOutOfBoundsException("Error"); ------ 1

            return true;
      }

      public Flat mostExpensiveFlat(int f) { ---------- /9
            RentalPlace maxFlat = null;  ---------- 1

            for (int i = 0; i < nbPlaces; i++) { ---------- 1
               if (arPlaces[i] instanceof Flat && ---------- 1
                   arPlaces[i].getFloor() == f) {  ---------- 1
                        if ((maxFlat == null) || ---------- 1
                             (arPlaces[i].calculateRent() > maxFlat.calculateRent()))-- 1
                               maxFlat = arPlaces[i]; ---------- 1
                   }
            }

            return (Flat) maxFlat; ---------- 1+1
      }
```

```
public Flat[] getFlatsWithLounge(int n) {  ---------- /10
      Flat[] flats = new Flat[nbPlaces];  ---------- 1
      Flat f;
      int j = 0; ---------- 1

      for (int i = 0; i < nbPlaces; i++) { ---------- 1
            if (arPlaces[i] instanceof Flat) { ---------- 1
                  f = (Flat) arPlaces[i]; ---------- 1
                  if (f.getNbRooms() >= n && ---------- 1
                      f.hasLouange()==true) { ---------- 1
                        flats[j] = f; ---------- 1
                        j++; ---------- 1
                  }
            }
      }
      return flats; ---------- 1
}

public int readShops(String fileName, int lfs) throws IOException, /11
                                                    ClassNotFoundException {
      File f1 = new File(fileName);
      FileInputStream fo1 = new FileInputStream(f1); ---------- 0.5
      ObjectInputStream pf = new ObjectInputStream(fo1); ---------- 0.5

      RentalPlace p;
      int nb = 0;  ---------- 1
      try { ---------- 1
        while (true) {  ---------- 1
            p = (RentalPlace) pf.readObject();---------- 0.5 + 0.5
            if (  p instanceof Shop && ---------- 1
                  ((Shop) p).getFrontSide() > lfs) {  ------ 0.5 + 0.5
                        try {
                              addRentalPlace(p);  ---------- 1
                              nb++;  ---------- 1
                        } catch (IllegalArgumentException e) { }
            }
         }
      }
      catch(EOFException eof) {           }  ---------- 0.5
      catch(IndexOutOfBoundsException e) {           }  -------- 0.5
      pf.close();
      return nb;  ---------- 1
}
public void saveFlats(String fileName, double a) throws IOException {/5
      File f1 = new File(fileName);
      FileOutputStream fo1 = new FileOutputStream(f1);  ---------- 0.5
      ObjectOutputStream pf = new ObjectOutputStream(fo1);  ------- 0.5

      for (int i=0; i < nbPlaces; i++) {  ---------- 1
          if (arPlaces[i]instanceof Flat && ---------- 1
              arPlaces[i].getArea() > a ) ---------- 1
                  pf.writeObject(arPlaces[i]); ---------- 1
      }
      pf.close();
```

```
        }
}
```

Exercise 4:

Let's consider the same class ***Building*** described in exercise 3.



BuildingList class***:***

- Methods:
  - ***BuildingList ()***: constructor
  - ***insertAtFront (b: Building*)*:** this method adds the Building ***b*** at the beginning of the list.
  - ***countBuildingsOf (o: String*)*:** this method counts and returns the number of buildings that belong to the owner ***o***.
  - ***expensiveBuildings(f:*** int, ***r:*** double): This method returns a linked list containing all buildings where the rent of the most expensive flat in the floor ***f*** is greater than ***r***.

QUESTION: Translate into Java code the class ***BuildingList.***

Solution

```
public class BuildingList {  ---------- /20
      private Node head;  ---------- 1

      public BuildingList() {
            head = null;  ---------- 1
      }

      public void insertAtFront(Building b) {  ---------- /4
            Node current = new Node();  ---------- 1
            current.setData(b);  ---------- 1
            current.setNext(head);  ---------- 1
            head = current;  ---------- 1
      }

      public int countBuildings(String o) {  ---------- /7
            Node current = head;  ---------- 1
            int nb = 0;  ---------- 1

            while (current != null) {  ---------- 1
                  if (current.getData().getOwnerName().equals(o))  -------- 1
                        nb++;  ---------- 1

                  current = current.getNext();  ---------- 1
            }
            return nb;  ---------- 1
      }

      public BuildingList expensiveBuildings(int f, double r) {  ------- /7
            BuildingList res = new BuildingList();  ---------- 1
            Node current = head; ---------- 1

            while (current != null) {  ---------- 1
              if (current.getData().mostExpensiveFlat(f).calculateRent()>r) ---- 1
                        res.insertAtFront(current.getData());---------- 1

               current = current.getNext();  ---------- 1
            }
            return res;  ---------- 1
      }
}
```