Exercises 4.1 – 4.8
Unions of Classes and Methods

Exercise 4.1

Design a data representation for this problem:

. . . Develop a “bank account” program. The program keeps track of the balances in a person’s bank accounts. Each account has an id number and a customer’s name. There are three kinds of accounts: a checking account, a savings account, and a certificate of deposit (CD). Checking account information also includes the minimum balance. Savings account includes the interest rate. A CD specifies the interest rate and the maturity date. Naturally, all three types come with a current balance. . . .

- Represent the following examples using your classes:
  - Earl Gray, id# 1729, has $1,250 in a checking account with minimum balance of $500;
  - Ima Flatt, id# 4104, has $10,123 in a certificate of deposit whose interest rate is 4% and whose maturity date is June 1, 2005;
  - Annie Proulx, id# 2992, has $800 in a savings account; the account yields interest at the rate of 3.5%.

Exercise 4.2

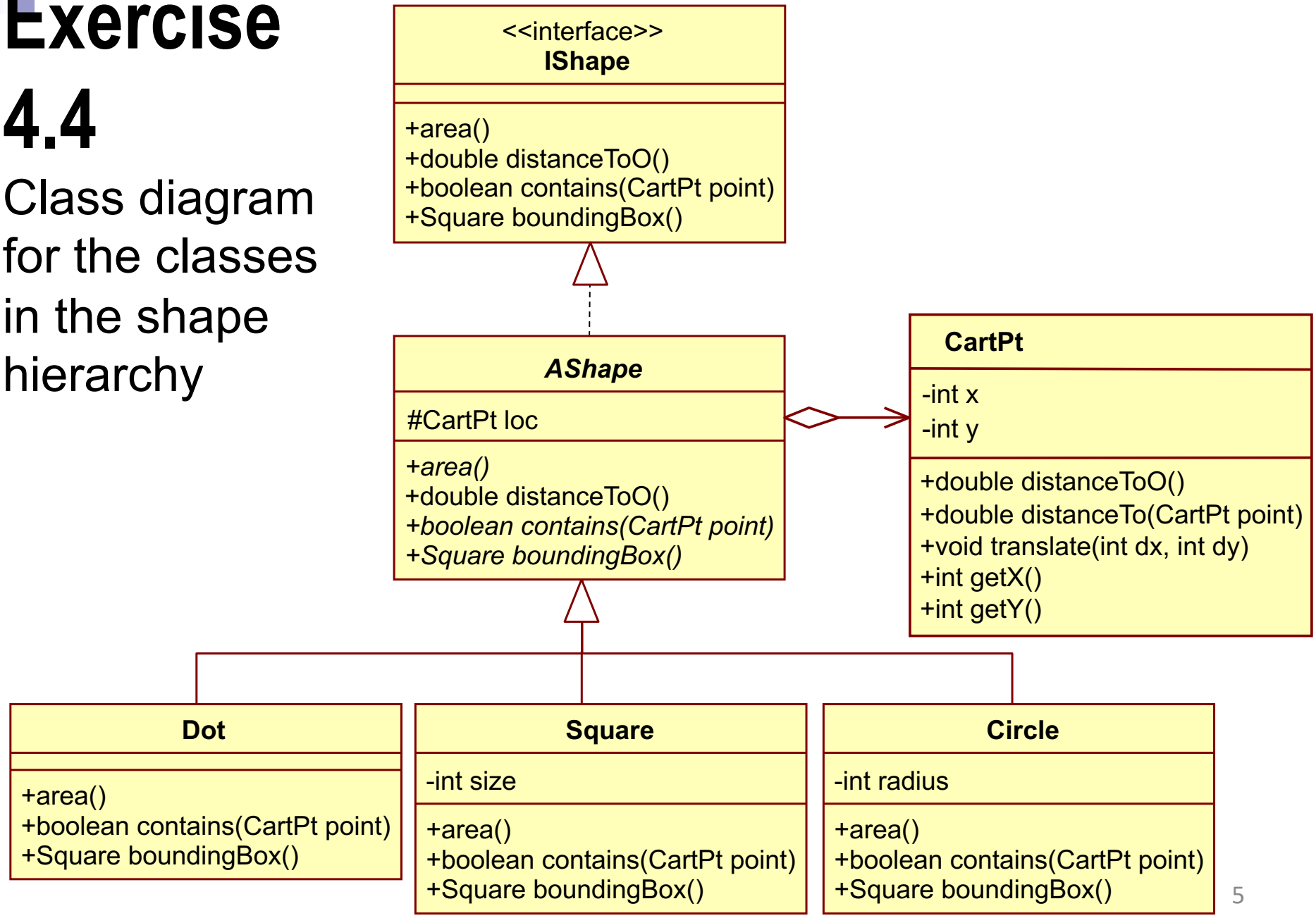
- Develop a program that creates a gallery from three different kinds of records: images (gif), texts (txt), and sounds (mp3). All have names for source files and sizes (number of bytes). Images also include information about the height, the width, and the quality of the image. Texts specify the number of lines needed for visual representation. Sounds include information about the playing time of the recording, given in seconds.
- Examples:
  - an image, stored in flower.gif; size: 57,234 bytes; width: 100 pixels; height: 50 pixels; quality: medium;
  - a text, stored in welcome.txt; size: 5,312 bytes; 830 lines;
  - a music piece, stored in theme.mp3; size: 40960 bytes, playing time 3 minutes and 20 seconds.

4.3 Extended exercises

- The administrators of metropolitan transportation agencies manage fleets of vehicles.
  Develop data definitions for a collection of such vehicles. The collection should include at least buses, limos, cars, and subways. Add at least two attributes per class of vehicle.

Exercise 4.4

Class diagram for the classes in the shape hierarchy



Exercise 4.4 con't

4.4.1 Design an extension for the class hierarchy of shapes that deals with `Rectangles`.
The extension should cope with all the abstract methods in `AShape`

4.4.2 Design an extension for the class hierarchy of shapes so that a program can request the length of the perimeter for each shape in the hierarchy

4.4.3 Extended
Compute the bounding box the class hierarchy of shapes that deals with `Rectangles`. The extension should cope with all the abstract methods in `AShape`

Exercise 4.5

- Develop a program that creates a gallery from three different kinds of records: images (gif), texts (txt), and sounds (mp3). All have names for source files and sizes (number of bytes). Images also include information about the height, the width, and the quality of the image. Texts specify the number of lines needed for visual representation. Sounds include information about the playing time of the recording, given in seconds.
- Develop the following methods for this program:
  - **timeToDownload**, which computes how long it takes to download a file at some network connection speed, typically given in bytes per second;
  - **smallerThan**, which determines whether the file is smaller than some given maximum size that can be mailed as an attachment;
  - **sameName**, which determines whether the name of a file is the same as some given name.

Exercise 4.6

Develop a program that keeps track of the items in the **grocery store**. For now, assume that the store deals only with ice cream, coffee, and juice. Each of the items is specified by its brand name, weight (gram) and price (cents). Each coffee is also labeled as either regular or decaffeinated. Juice items come in different flavors, and can be packaged as frozen, fresh, bottled, or canned. Each package of ice cream specifies its flavor and whether this is a sorbet, a frozen yogurt, or regular ice cream.

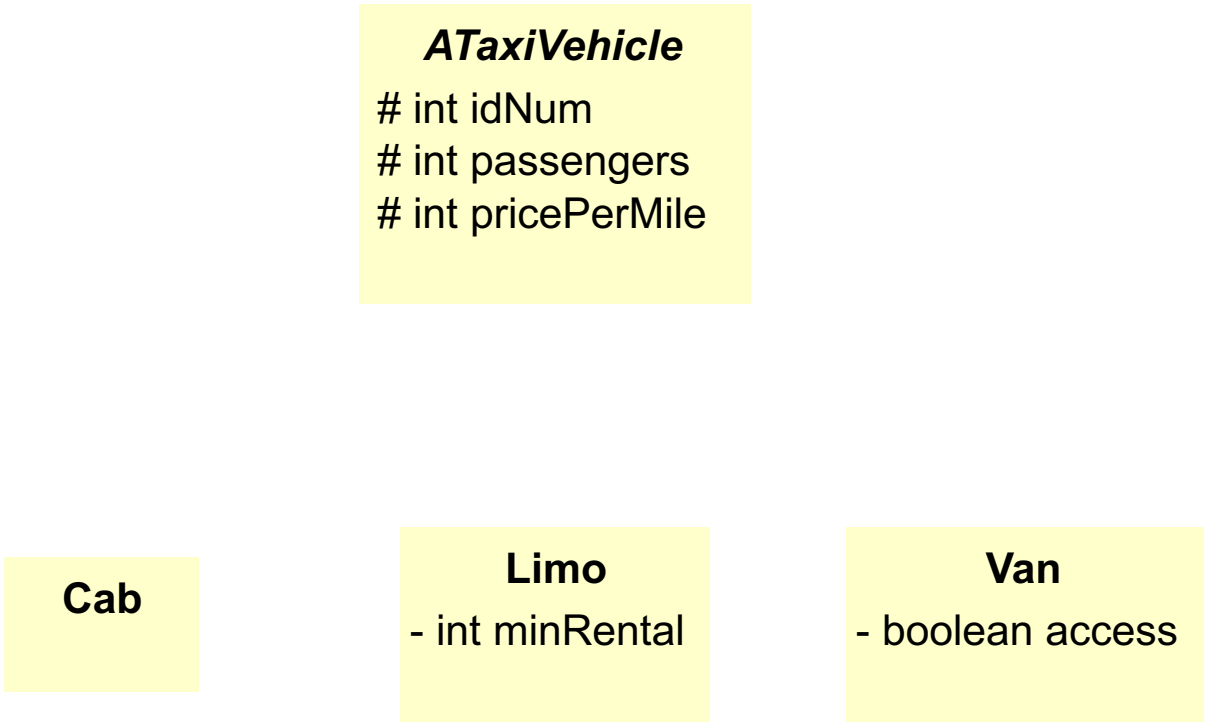
Design the following methods:

- **unitPrice**, which computes the unit price (cents per gram) of some grocery item;
- **lowerPrice**, which determines whether the unit price of some grocery item is lower than some given amount;
- **cheaperThan**, which determines whether a grocery item is cheaper than some other, given one in terms of the unit cost.

Solutions

Exercise 4.7

- Recall the class hierarchies concerning taxi vehicles from exercise 4.2:



Exercise 4.7 (cont)

Add the following methods to the appropriate classes in the hierarchy:

- *fare*, which computes the fare for a vehicle, based on the number of miles traveled, and using the following formulas for different vehicles:
  - passengers in a cab just pay flat fee per mile
  - passengers in a limo must pay at least the minimum rental fee, otherwise they pay by the mile
  - passengers in a van pay $1.00 extra for each passenger
- *lowerPrice*, which determines whether the fare for a given number of miles is lower than some amount;
- *cheaperThan*, which determines whether the fare in one vehicle is lower than the fare in another vehicle for the same number of miles.

Solutions

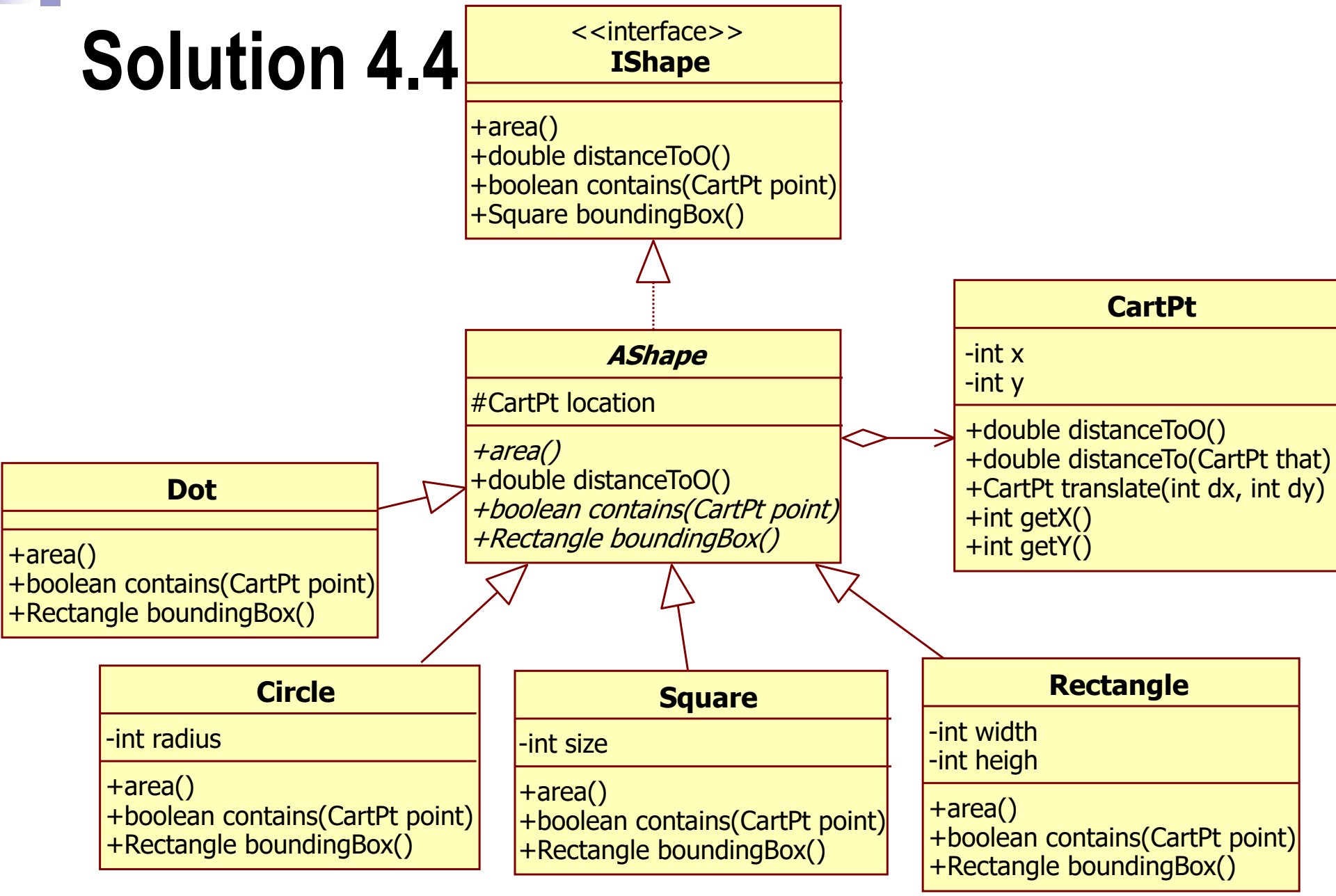
Exercise 4.8

- Develop a program that assists a bookstore manager in a discount bookstore. The program should keep a record for each book. The record must include its title, the author's name, its price, and its publication year. In addition, there are three kinds of books with different pricing policy. The hardcover books are sold at 20% off. The sale books are sold at 50% off. The paperbacks are sold at the list price. Here are your tasks:
- Develop a class hierarchy for representing books in the discount bookstore.
- Develop the following methods:
  - ***salePrice***, which computes the sale price of each book;
  - ***cheaperThan***, which determines whether a book is cheaper than another book;
  - ***sameAuthor***, which determines whether a book was written by some given author which wrote another book.

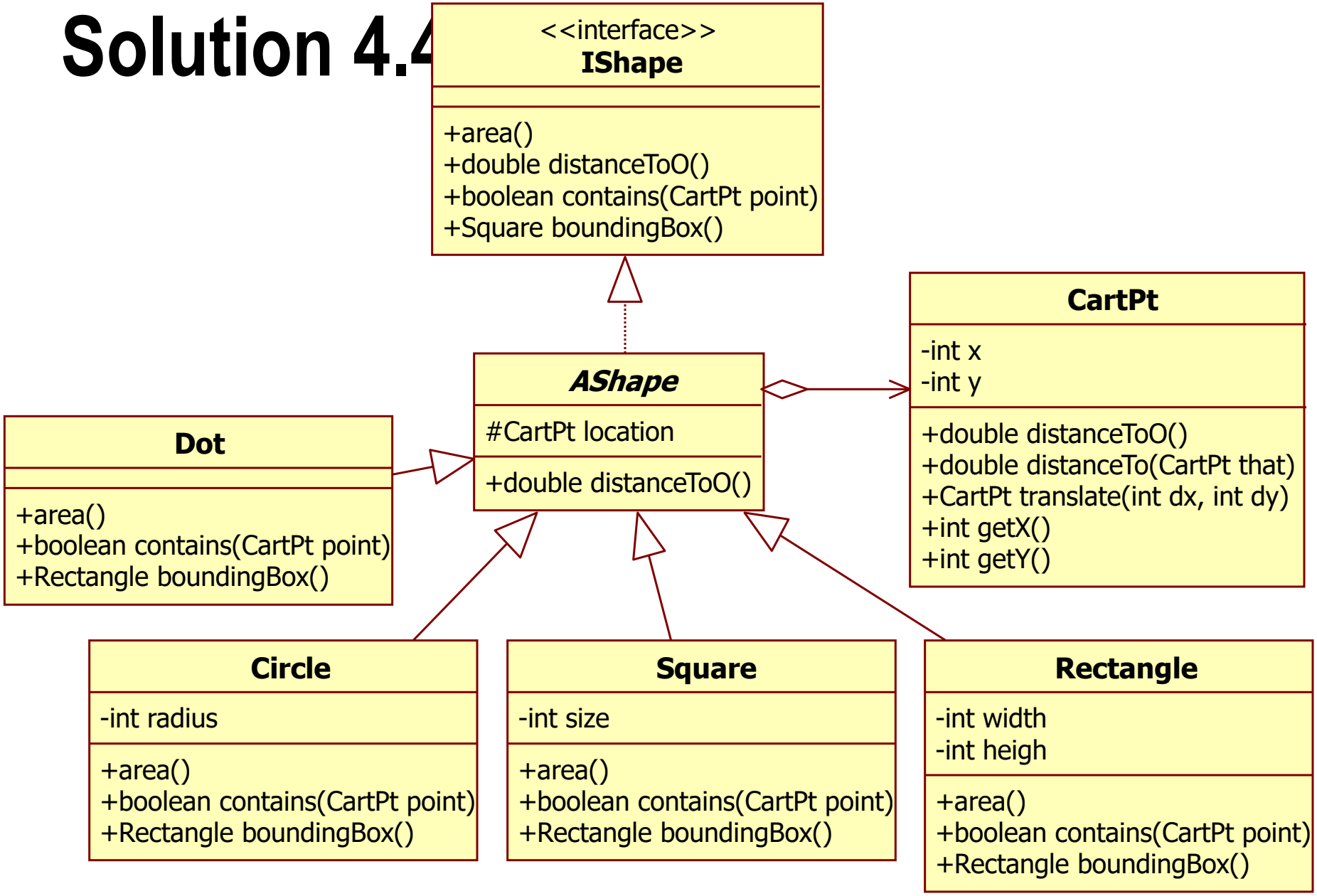
Solutions

Relax &
…Do Exercises …

Solution 4.4


<<interface>>
IShape
+area()
+double distanceToO()
+boolean contains(CartPt point)
+Square boundingBox()
AShape
#CartPt location
+area()
+double distanceToO()
+boolean contains(CartPt point)
+Rectangle boundingBox()
CartPt
-int x
-int y
+double distanceToO()
+double distanceTo(CartPt that)
+CartPt translate(int dx, int dy)
+int getX()
+int getY()
Dot
+area()
+boolean contains(CartPt point)
+Rectangle boundingBox()
Circle
-int radius
+area()
+boolean contains(CartPt point)
+Rectangle boundingBox()
Square
-int size
+area()
+boolean contains(CartPt point)
+Rectangle boundingBox()
Rectangle
-int width
-int heigh
+area()
+boolean contains(CartPt point)
+Rectangle boundingBox()

Solution 4.4


<<interface>>
IShape
+area()
+double distanceToO()
+boolean contains(CartPt point)
+Square boundingBox()
CartPt
-int x
-int y
+double distanceToO()
+double distanceTo(CartPt that)
+CartPt translate(int dx, int dy)
+int getX()
+int getY()
AShape
#CartPt location
+double distanceToO()
Dot
+area()
+boolean contains(CartPt point)
+Rectangle boundingBox()
Circle
-int radius
+area()
+boolean contains(CartPt point)
+Rectangle boundingBox()
Square
-int size
+area()
+boolean contains(CartPt point)
+Rectangle boundingBox()
Rectangle
-int width
-int heigh
+area()
+boolean contains(CartPt point)
+Rectangle boundingBox()

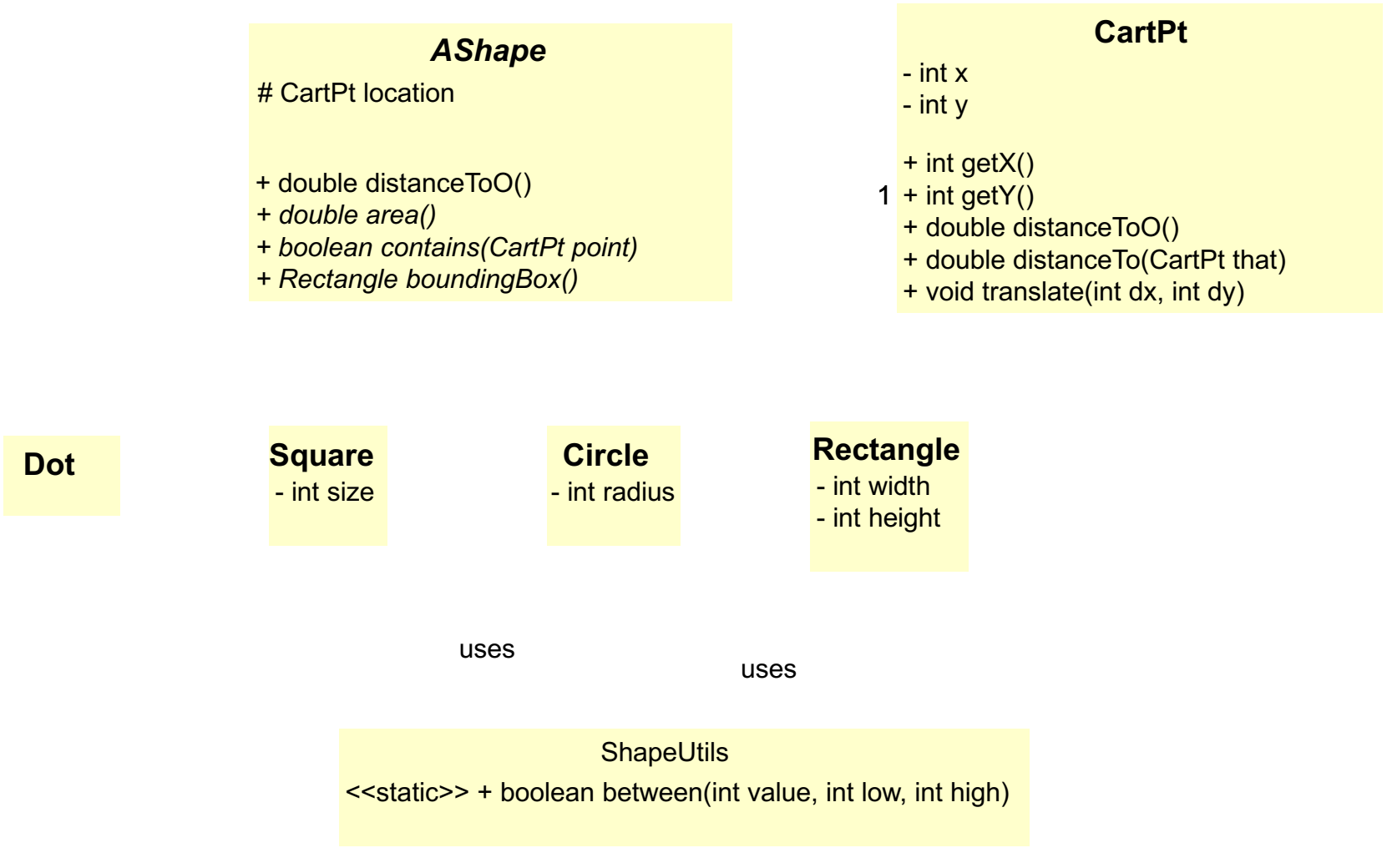
Solution 4.4.1: `ShapeUtils`

Q: Both **`Square`** and **`Rectangle`** do need the method **`between()`**. Remove similarities in these two classes

A: Since **`between()`** could exist independently, that it does not need to belong to any of the existing classes, it could be put in a helping class

```
public class ShapeUtils {
   public static boolean between(int value, int low, int high) {
      return (low <= value) && (value <= high);
   }
}
```

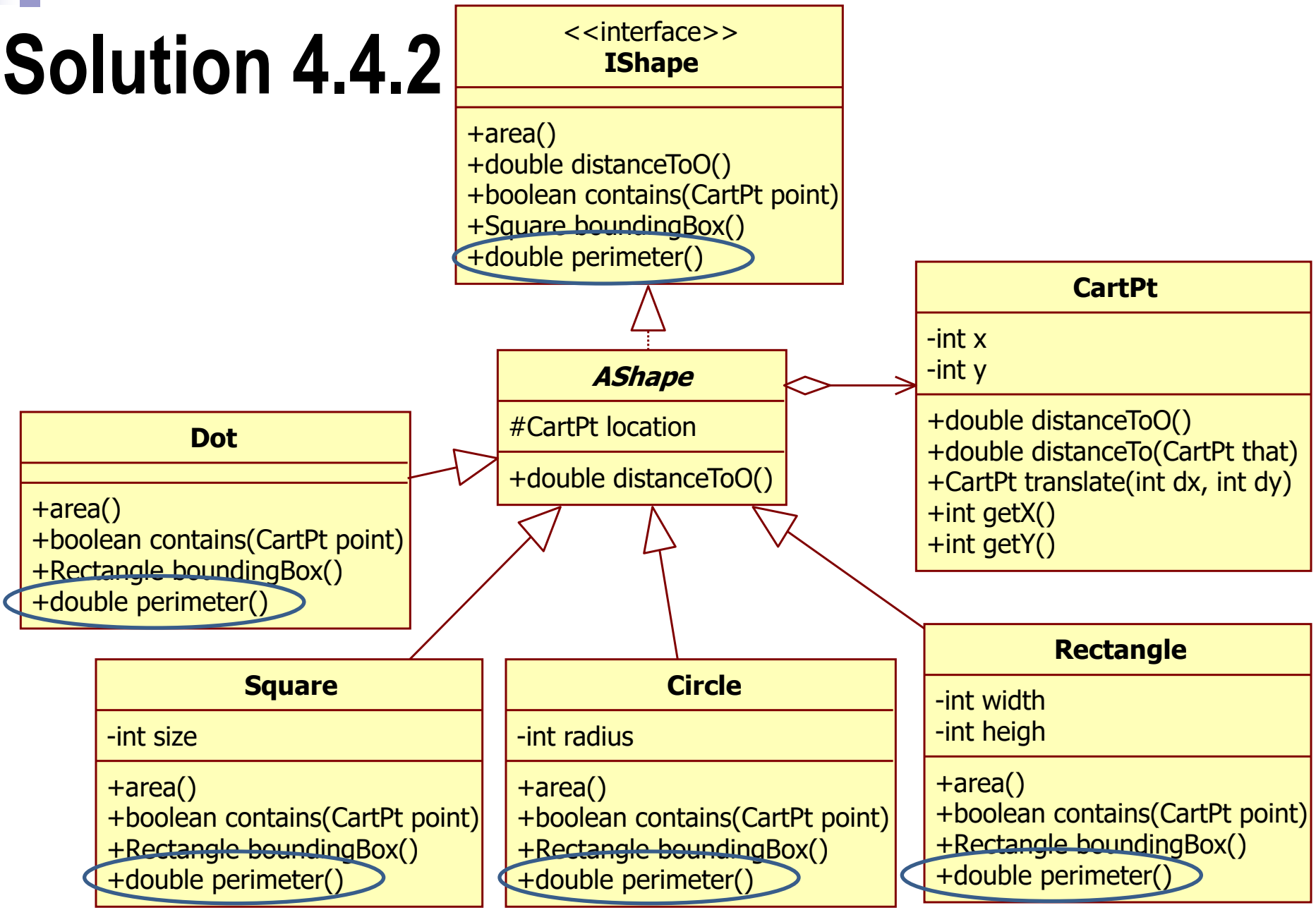
Solution 4.4.1: Improved Design



Solution 4.4.1: Rectangle

```
public class Rectangle extends AShape {
   private int width;
   private int height;

   public Rectangle(CartPt location, int width, int height) {
      super(location);
      this.width = width; this.height = height;
   }

   public double area() {
      return this.width * this.height;
   }

   public boolean contains(CartPt point) {
      int thisX = this.location.getX();
      int thisY = this.location.getY();
      return ShapeUtils.between(point.getX(), thisX, thisX + this.width)
         && ShapeUtils.between(point.getY(), thisY, thisY + this.height);
   }

   public Rectangle boundingBox() {
      return new Rectangle(this.location, width, height);
   }
}
```

Solution 4.4.2



Solution 4.4.2 (cont.)

```
// in Dot
public double perimeter() {
   return 1.0;
}
```

```
// in Square
public double perimeter() {
   return this.size * 4;
}
```

```
 // in Rectangle
public double perimeter() {
   return (this.width + this.height) * 2;
}
```

```
// in Circle
public double perimeter() {
   return this.radius * 2 * Math.PI;
}
```

Solution 4.5: A gallery Class Diagram

AGallery

String fileName
int fileSize // number of bytes

+ double timeToDownLoad(double networkSpeed)
+ boolean smallerThan(int maximumSize)
+ boolean sameName(String givenName)

Image

- int height
- int width
- String quality

Text

- int numberOfLines

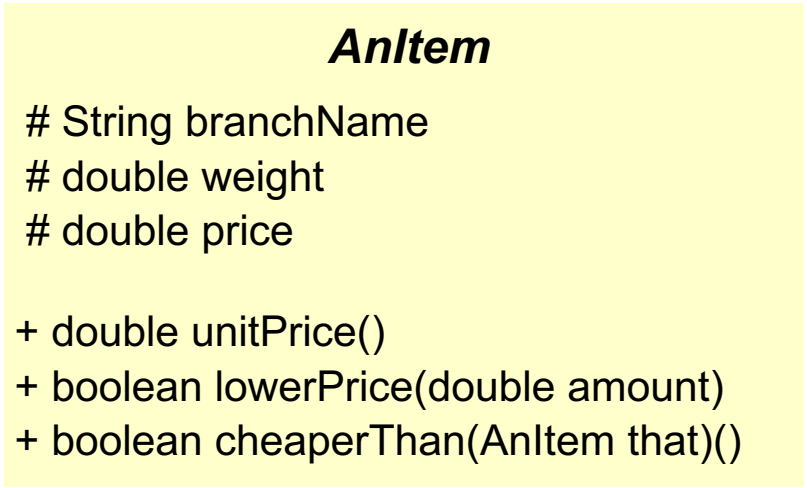
Sound

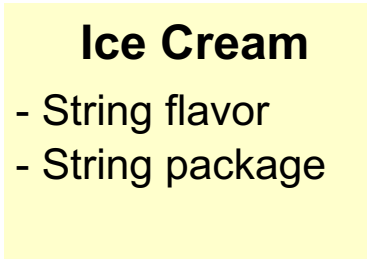
- int playingTime // in seconds

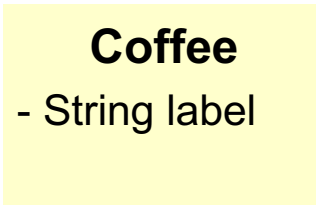
Solution 4.5

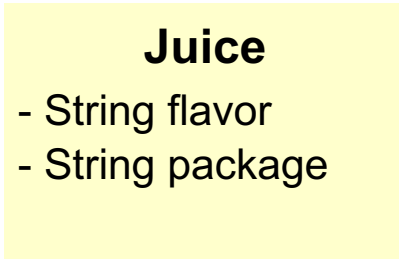
```
public abstract class AGallery {
    protected String fileName;
    protected int fileSize;
    public AGallery(String filename, int fileSize) {
        this.filename = fileName;
        this.fileSize = fileSize;
    }
    public double timeToDownLoad(double networkSpeed) {
        return this.fileSize/ networkSpeed;
    }

    public boolean smallerThan(int maximumSize){
        return this.fileSize < maximumSize;
    }

    public boolean sameName(String givenName){
        return this.fileName.equals(givenName);
    }
}
```

Solution 4.6: Class Diagram









Solution 4.6

```
public abstract class AnItem {
   protected String branchName;
   protected double weight;
   protected double price;
   public AnItem(String branchName, double weight, double price) {
      this.branchName = branchName;
      this.weight = weight;
      this.price = price;
   }

   public double unitPrice() {
      return this.price / this.weight;
   }
   public boolean lowerPrice(double amount) {
      return this.unitPrice() < amount;
   }
   public boolean cheaperThan(AnItem that) {
      return this.unitPrice() < that.unitPrice();
   }
}
```

Ice Cream

```
public class IceCream extends AnItem {
    private String flavor;
    private String package;

    public IceCream(String branchName, double weight,
                double price, String flavor, String package) {
        super(branchName, weight, price);
        this.flavor = flavor;
        this.package = package;
    }
}
```

Coffee

```
public class Coffee extends AnItem {
   private String label;

   public Coffee(String label, String branchName,
                 double weight, double price) {
      super(branchName, weight, price);
      this.label = label;
   }
}
```

Juice

```
public class Juice extends AnItem {
   private String flavor;
   private String package;

   public Juice(String branchName, double weight,
                double price, String flavor,
                String package) {
      super(branchName, weight, price);
      this.flavor = flavor;
      this.package = package;
   }
}
```

back

Solution 4.7: Class Diagram

ATaxiVehicle

int idNum
int passengers
int pricePerMile

+ *double fare(double numberOfMiles)*
+ boolean lowerPrice(double numberOfMiles, double amount)
+ boolean cheaperThan(double numberOfMiles, ATaxiVehicle that)

Cab

+ double fare(double numberOfMiles)

Van

- boolean access

+ double fare(double numberOfMiles)

Limo

- int minRental

+ double fare(double numberOfMiles)

Solution 4.7

```
public abstract class ATaxiVehicle {
   protected int idNum;
   protected int passengers;
   protected int pricePerMile;
   public ATaxiVehicle(int idNum, int passengers, int pricePerMile) {
      this.idNum = idNum;
      this.passengers = passengers;
      this.pricePerMile = pricePerMile;
   }

   public abstract double fare(double numberOfMiles);

   public boolean lowerPrice(double numberOfMiles, double amount) {
      return this.fare(numberOfMiles) < amount;
   }

   public boolean cheaperThan(double numberOfMiles, ATaxiVehicle that) {
      return this.fare(numberOfMiles) < that.fare(numberOfMiles);
   }
}
```

Cab

```
public class Cab {
   public Cab(int idNum, int passengers, int pricePerMile) {
      super(idNum,passengers, pricePerMile);
   }

   public double fare(double numberOfMiles) {
      return this.pricePerMile * numberOfMiles;
   }
}
```

Limo

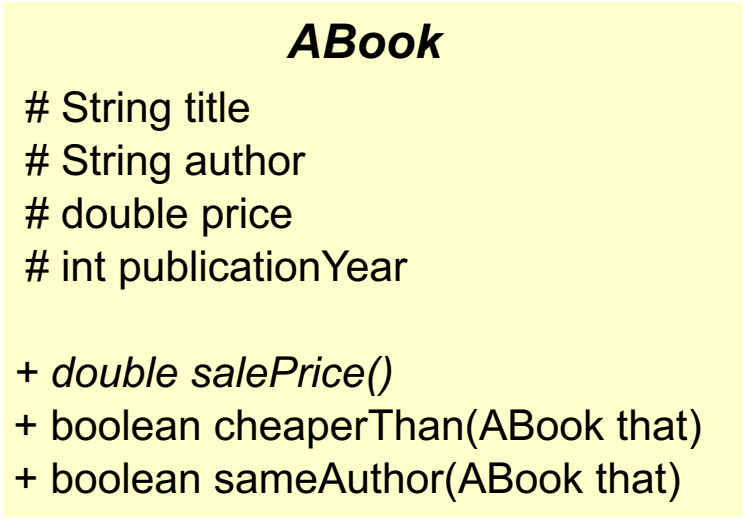
```
public class Limo extends ATaxiVehicle {
   private int minRental;

   public Limo (int minRental, int idNum,
                int passengers, int pricePerMile) {
      super(idNum,passengers, pricePerMile);
      this.minRental = minRental;
   }

   public double fare(double numberOfMiles) {
      if (this.pricePerMile * numberOfMiles < minRental)
         return this.minRental;
      else
         return this.pricePerMile * numberOfMiles;
   }
}
```

Van

```
public class Van extends ATaxiVehicle {
   private boolean access;

   public Van(boolean access, int idNum,
              int passengers, int pricePerMile) {
      super (idNum,passengers, pricePerMile);
      this.access = access;
   }

   public double fare(double numberOfMiles) {
      return
         this.pricePerMile * numberOfMiles + this.passengers;
    }
}
```

Back

4.8 Class Diagram



4.8 Solution

```
public abstract class ABook {
   protected String title;
   protected String author;
   protected double price;
   protected int publicationYear;
   public ABook(String title, String author,
                double price, int publicationYear){
      this.title =title;
      this.author = author;
      this.price = price;
      this.publicationYear = publicationYear;
   }

   public abstract double salePrice();
   public boolean cheaperThan(ABook that){
      return this.salePrice() < that.salePrice();
   }
   public boolean sameAuthor(ABook that){
      return this.author.equals(that.author);
   }
}
```

Hardcover Book

```
public class Hardcover extends ABook {
   public Hardcover(String title, String author,
                    double price, int publicationYear) {
      super (title, author, price, publicationYear);
   }

   public double salePrice()  {
      return this.price * 0.8;
   }
}
```

Sale Book

```
public class Sale extends ABook {
   public Sale(String title, String author,
               double price, int publicationYear) {
      super(title, author, price, publicationYear);
   }

   public double salePrice() {
      return this.price * 0.5;
   }
}
```

Paperback Book

```
public class Paperback extends Abook {
   public Paperback(String title, String author,
                    double price, int publicationYear) {
      super(title, author, price, publicationYear);
   }

   public double salePrice() {
      return this.price;
   }
}
```

Back