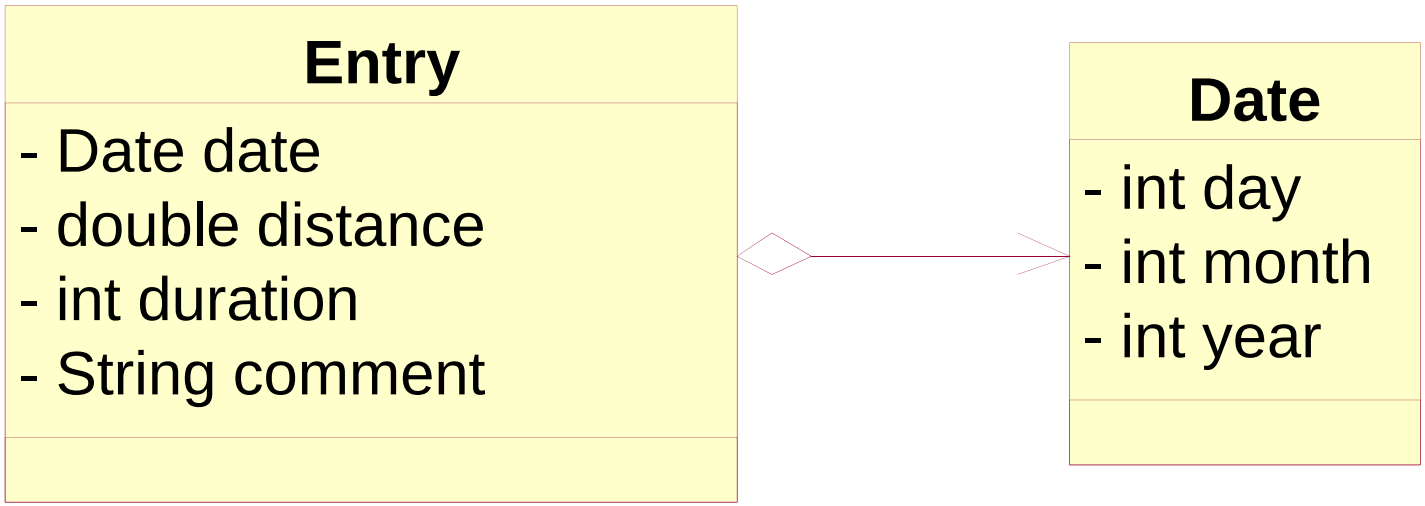
# Class References, Object Containment and Methods

# Runner's training log

- Develop a program that manages a runner's training log. Every day the runner enters one entry concerning the day's run. Each entry includes the day's date, the distance of the day's run, the duration of the run, and a comment describing the runner's post-run feeling.
- Examples:
  - on June 5, 2003: 5.3 miles in 27 minutes, feeling good;
  - on June 6, 2003: 2.8 miles in 24 minutes, feeling tired
  - on June 23, 2003: 26.2 miles in 150 minutes, feeling exhausted;

# Class Diagram



# Define class and constructor

```
public class Entry {
   private Date date;
   private double distance;
   private int duration;
   private String comment;
   public Entry(Date date, double distance, int duration,
         String comment) {
      this.date = date;
      this.distance = distanc
      this.duration = duratio
      this.comment = comment;
   }
}
```

contain

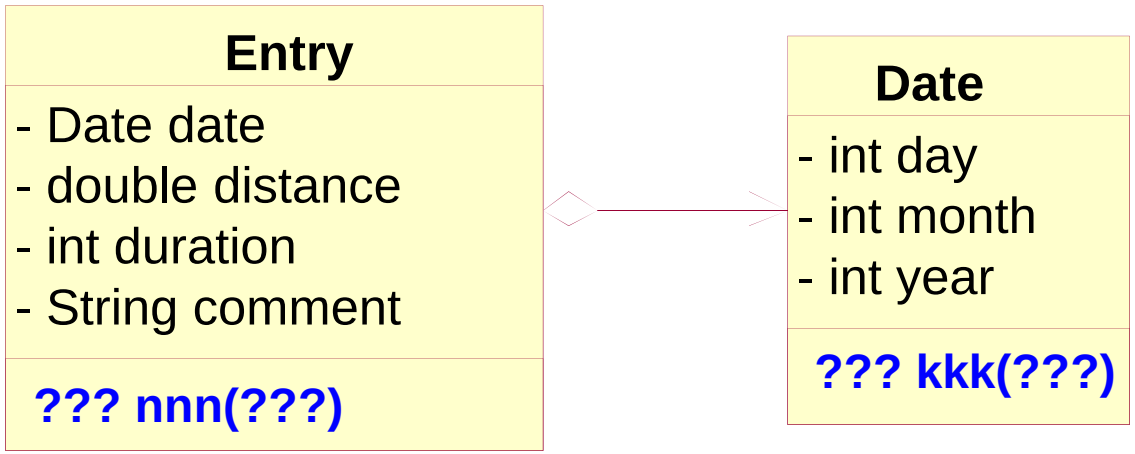
```
public class Date {
   private int day;
   private int month;
   private int year;
   public Date(int day, int month,
               int year) {
      this.day = day;
      this.month = month;
      this.year = year;
   }
}
```

# Test constructor

```
import junit.framework.*;
public class EntryTest extends TestCase {
   public void testDateContructor() {
      new Date(5, 6, 2004);
      Date date1 = new Date(6, 6, 2004);
      Date date2 = new Date(23, 6, 2004);
   }

   public void testEntryContructor() {
      new Entry(new Date(5, 6, 2004), 5.3, 27, "good");

      Date date1 = new Date(6, 6, 2004);
      new Entry(date1, 2.8, 24, "tired");

      Date date2 = new Date(23, 6, 2004);
      new Entry(date2, 26.2, 159, "exhausted");
   }
}
```

# Methods for containment

# Add methods to the `Entry`



# Java template for `Entry`

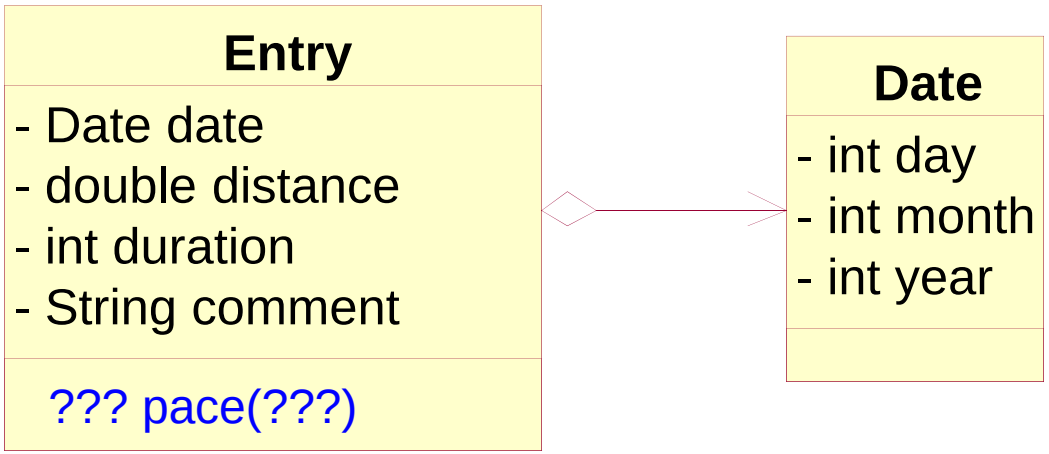
```
public class Entry {
   private Date date;
   private double distance;
   private int duration;
   private String comment;
   public Entry(Date date, double distance, int duration,
         String comment) {
      this.date = date;
      this.distance = distance;
      this.duration = duration;
      this.comment = comment;
   }

   public ??? nnn(???) {
      ...this.date.kkk(???)...
      ...this.distance...this.duration...this.comment...
   }
}
```

# Java template for `Date`

```
public class Date {
   private int day;
   private int month;
   private int year;
   public Date(int day, int month,
               int year) {
      this.day = day;
      this.month = month;
      this.year = year;
   }

   public ??? kkk(???) {
      ...this.day...
      ...this.month...
      ...this.year...
   }
}
```

# Computes the pace for a daily entry

- For each entry, the program should compute how fast the runner ran in *minutes per mile*.
  ... Develop a method that computes the pace for a daily entry.



# Design `pace()` method

Purpose and contract (method signature)

```
// computes the pace for a daily entry
public double pace() {
}
```

- Examples
  - `new Entry(new Date(5, 6, 2004), 5.3, 27, "good").pace()` should produce **5.094**
  - `new Entry(new Date(6, 6, 2004), 2.8, 24, "tired").pace()` should produce **8.571**
  - `new Entry(new Date(23, 6, 2004), 26.2, 159, "exhausted").pace()` should produce **6.069**

# Design `pace()` method (con't)

Template

```
// computes the pace for a daily entry
public double pace() {
   ...this.date...
   ...this.duration...
   ...this.distance...
   ...this.comment...
}
```

Implement

```
public double pace() {
   return this.duration / this.distance;
}
```

# Design `pace()` method (con't)

- Unit testing

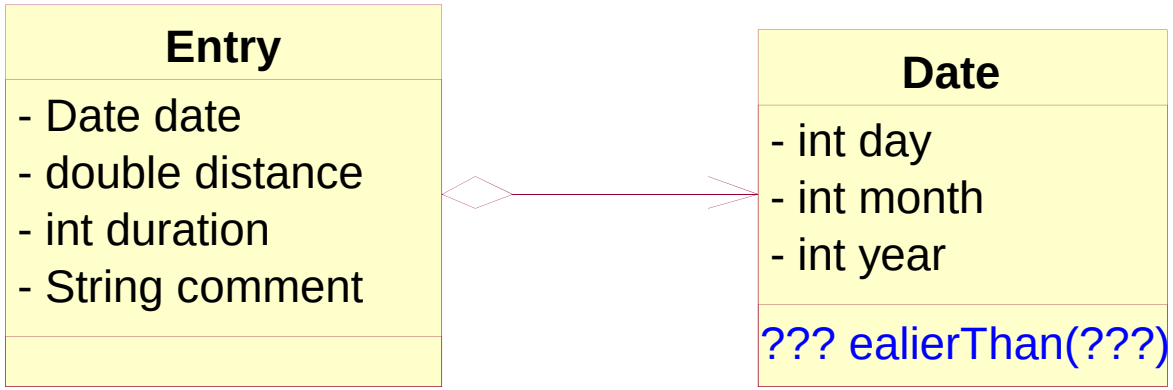
```
public void testPace() {
   Entry entry1 = new Entry(new Date(5, 6, 2004), 5.3, 27, "good");
   assertEquals(entry1.pace(), 5.094, 0.001);

   Entry entry2 = new Entry(new Date(6, 6, 2004), 2.8, 24, "tired");
   assertEquals(entry2.pace(), 8.571, 0.001);

   Entry entry3 = new Entry(new Date(23, 6, 2004), 26.2, 159, "exhausted");
   assertEquals(entry3.pace(), 6.069, 0.001);
}
```

# Compare `Date`: early than

- A runner's log refers to Dates and a natural question concerning comparing dates is when one occurs earlier than another one.
  Develop a method that determines whether one date occurs earlier than another date.
- Hint:
  - The first possibility is that the first date is in the year preceding the other.
  - Next, if the years are the same, the month in the first date is before the month in the second date.
  - Finally, if both the year and the month values are the same, the date in the first date is before the day in the second date.

# Delegation

- **Q**: Which class (**`Entry`** or **`Date`**) should we put **`ealierThan()`** method in ?
- **A**: The **`ealierThan()`** method deals with properties of the **`Date`** so that we delegate this computational task to the corresponding methods in **`Date`** class



# Design `earlierThan()` method

- Purpose and contract (method signature)

```
// is this date early than the other date
public boolean earlierThan(Date that)
```

- Examples
  - `new Date(30, 6, 2003).earlierThan(new Date(1, 1, 2004))` should produce **`true`**
  - `new Date(1, 1, 2004).earlierThan(new Date(1, 12, 2003))` should produce **`false`**
  - `new Date(15, 12, 2004).earlierThan(new Date(31, 12, 2004))` should produce **`true`**

# Design `earlyThan()` method (con't)

Template

```
// is this date early than the other date
public boolean earlyThan(Date that) {
   ...this.day...this.month...this.year...
   ...that.day...that.month...that.year...
}
```

Implement

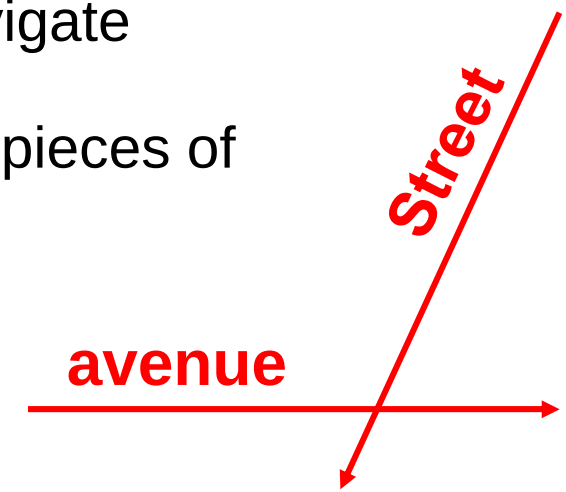
```
public boolean earlierThan(Date that) {
   if (this.year < that.year) return true;
   if (this.year > that.year) return false;
   if (this.month < that.month) return true;
   if (this.month > that.month) return false;
   if (this.day < that.day) return true;
   return false;
}
```

# Unit Testing

```
pubblic class EntryTest extends TestCase {
   ...
   public void testEarlierThan() {
      Date date1 = new Date(30, 6, 2003);
      Date date2 = new Date(1, 1, 2004);
      Date date3 = new Date(1, 12, 2004);
      Date date4 = new Date(15, 12, 2004);
      Date date5 = new Date(31, 12, 2004);

      assertTrue(date1.earlierThan(date2));
      assertTrue(date2.earlierThan(date3));
      assertTrue(date3.earlierThan(date4));
      assertTrue(date4.earlierThan(date5));

      assertFalse(date1.earlierThan(date1));
      assertFalse(date5.earlierThan(date4));
      assertFalse(date4.earlierThan(date3));
      assertFalse(date3.earlierThan(date2));
      assertFalse(date2.earlierThan(date1));
   }
}
```

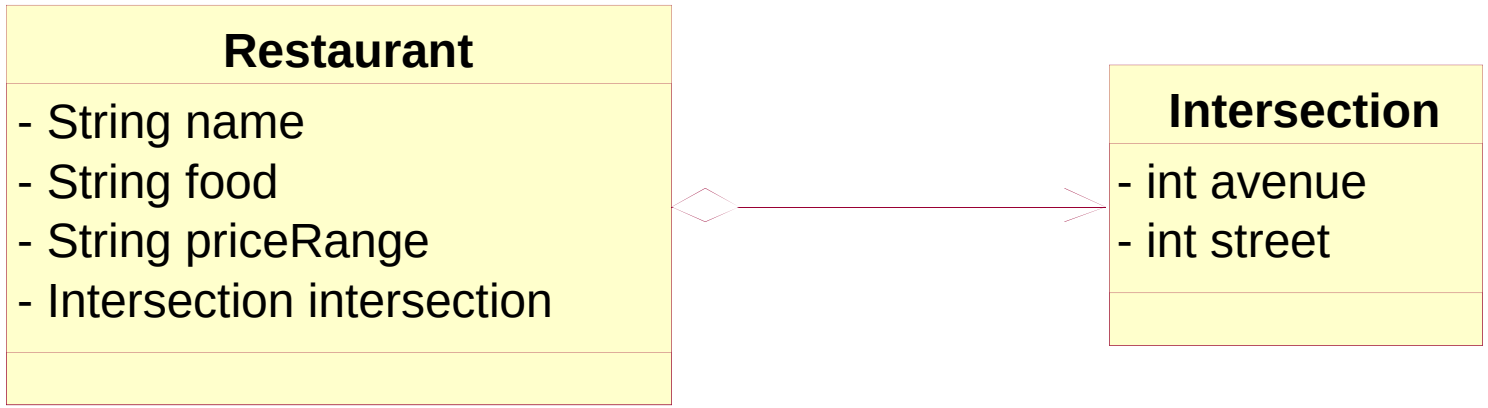
# Restaurant example

- Develop a program that helps a visitor navigate Manhattan's restaurant scene.
  The program must be able to provide four pieces of information for each restaurant: its name, the kind of food it serves, its price range, and the closest intersection (street and avenue).




- Examples:
  - La Crepe, a French restaurant, on 7th Ave and 65th Street, moderate;
  - Bremen Haus, a German restaurant on 2nd Ave and 86th Street, moderate;
  - Moon Palace, a Chinese restaurant on 10th Ave and 113th Street, inexpensive;

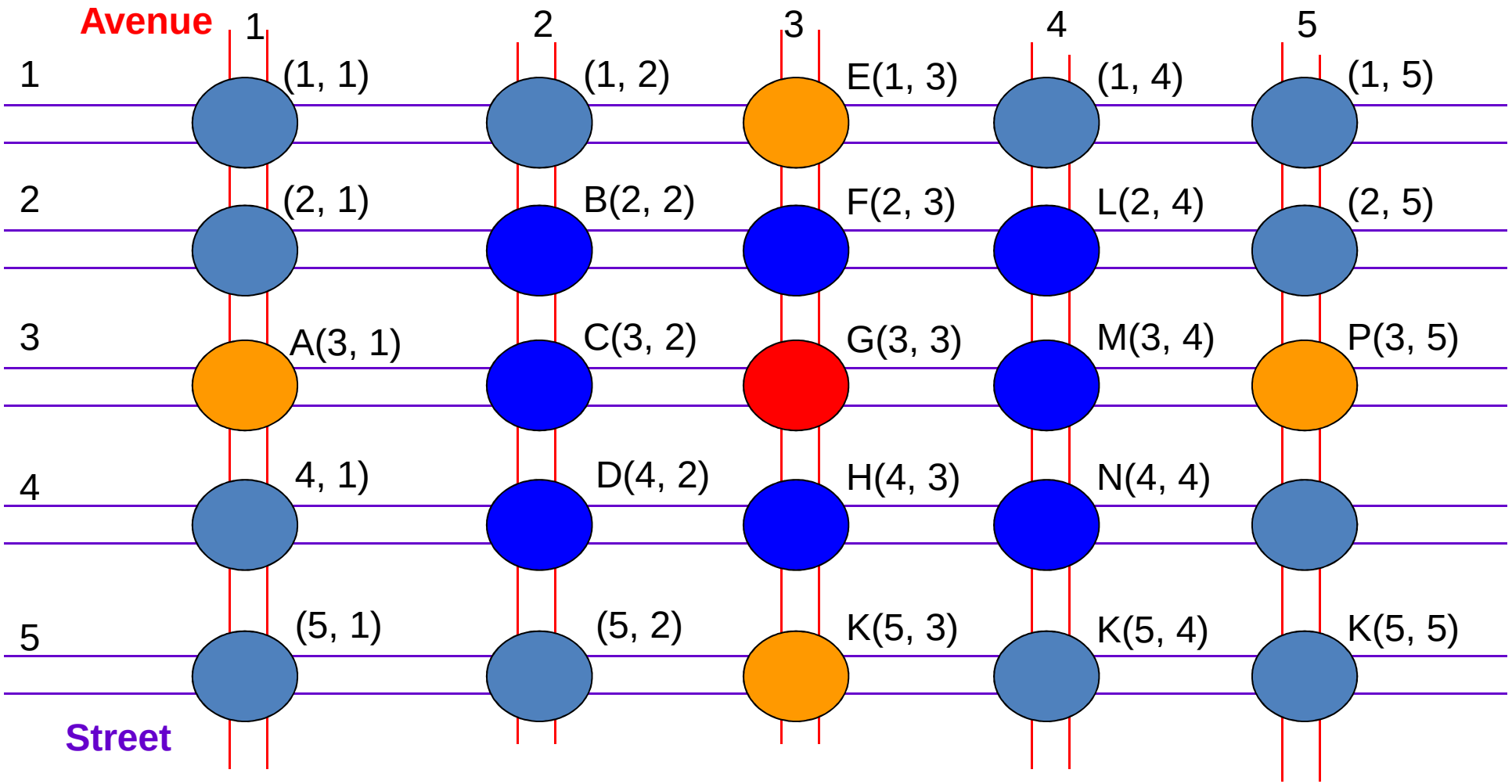
# Class Diagram



# Problem Statement

- Develop a method to help visitors to find out whether two restaurants are close to each other
- Two restaurants are "close" to each other if they are at most one avenue *and* at most one street away from each other

- **Q:** Add this method to the class diagram




X(Street, Avenue)

G “closes” B, C, D, F, H, L, M, N

The considered Intersection

Intersections “close” to the considered Intersection

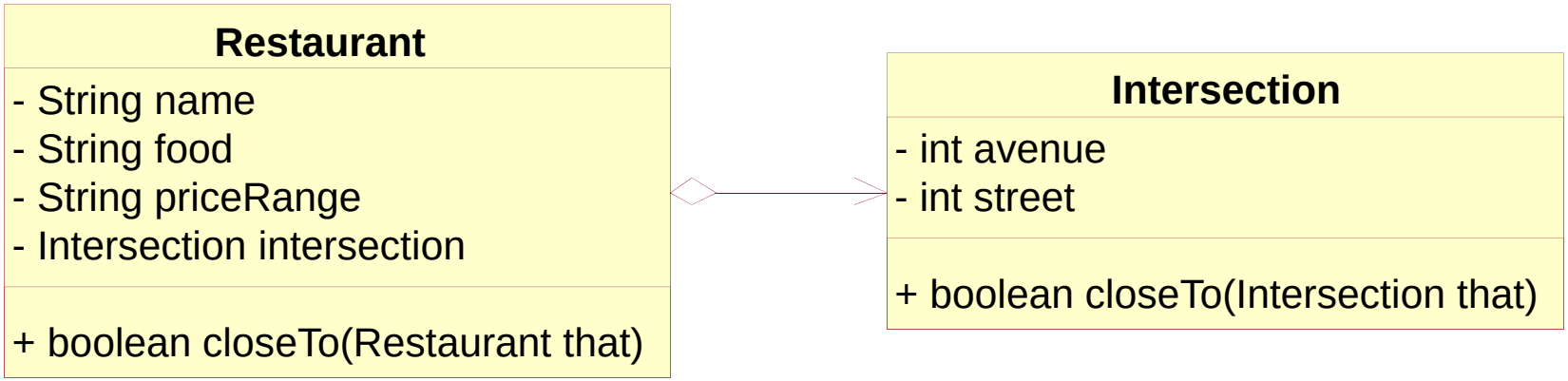
Intersections not “close” to the considered Intersection

# Delegation

**Q**: Which class (`Restaurant` or `Intersection`) should we put `closeTo()` method in ?

**A**: Put `closeTo()` in both classes.

- The `closeTo()` method deals with properties of the `Intersection` so that we delegate this computational task to the corresponding methods in `Intersection` class




**Q:** Create examples for the method `closeTo()` in the `Intersection` class

# Examples

```
Intersection i1 = new Intersection(3, 3);
Intersection i2 = new Intersection(3, 2);
i1.closeTo(i2); // should produce true
i1.closeTo(new Intersection(3, 5)); // should produce false
i2.closeTo(new Intersection(3, 5)); // should produce false

Restaurant r1 = new Restaurant("La Crepe", "French",
            "moderate", new Intersection(3, 3));
Restaurant r2 = new Restaurant("Das Bier", "German",
            "cheap", new Intersection(3, 2));
Restaurant r3 = new Restaurant("Sun", "Chinese",
            "cheap", new Intersection(3, 5));
r1.closeTo(r2);    // should produce true
r1.closeTo(r3);    // should produce false
r2.closeTo(r3);    // should produce false
```

# `closeTo` template in Intersection class

```
public class Intersection {
   private int avenue;
   private int street;
   public Intersection(int avenue, int street) {
      this.avenue = avenue;
      this.street = street;
   }

   // is this intersection close to another
   public boolean closeTo(Intersection that) {
      ...this.avenue...
      ...this.street...
      ...that.avenue...
      ...that.street...
   }
}
```

# `closeTo` template in Restaurant class

```
public class Restaurant {
   private String name;
   private String food;
   private String priceRange;
   private Intersection intersection;
   ...

   // is this restaurant close to another
   public boolean closeTo(Restaurant that) {
      ...this.name...this.food...
      ...this.priceRange... this.intersection...
      ...this.intersection.closeTo(...)...
      ...that.name... that.food...
      ...that.priceRange...that.intersection...
      ...that.intersection.closeTo(...)...
   }
}
```

# `closeTo` method implementation

```
public class Intersection {
   ...
   public boolean closeTo(Intersection that) {
      return ((Math.abs(this.avenue - that.avenue) <= 1) &&
              (Math.abs(this.street - that.street) <= 1));
   }
}
```

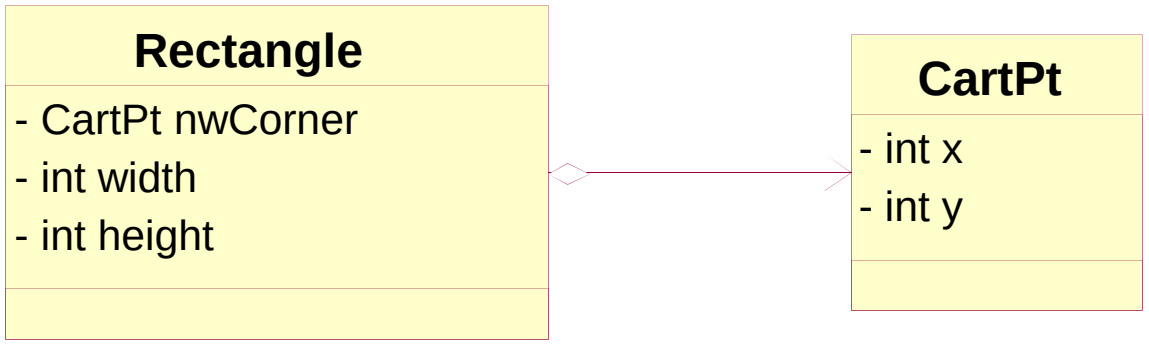
```
public class Restaurant {
   ...
   public boolean closeTo(Restaurant that) {
      return this.intersection.closeTo(that.intersection);
   }
}
```

# Unit Testing

```
public void testCloseTo() {
   Restaurant r1 = new Restaurant("La Crepe", "French", "moderate",
            new Intersection(3, 3));
   Restaurant r2 = new Restaurant("Das Bier", "German", "cheap",
            new Intersection(3, 2));
   Restaurant r3 = new Restaurant("Sun", "Chinese", "cheap",
            new Intersection(3, 5));

   assertTrue(r1.closeTo(r2));
   assertFalse(r1.closeTo(r3));
   assertFalse(r2.closeTo(r3));
}
```

# Rectangle example

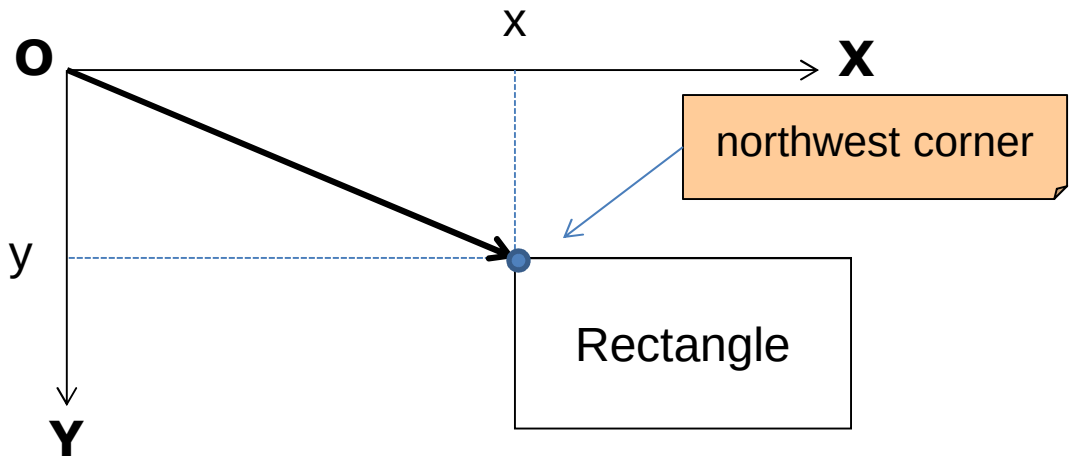
- The rectangles have *width*, *height and* are located on the Cartesian plane of a computer canvas, which has its origin in the northwest corner.



# Problem Statement

...Design a method that computes the distance of a **`Rectangle`** to the origin of the canvas.

- Considering that a **`Rectangle`** has many points, the meaning of this problem is clearly to determine the shortest distance of the **`Rectangle`** to the origin.
- This, in turn, means computing the distance between its northwest corner and the origin



# Problem Analysis

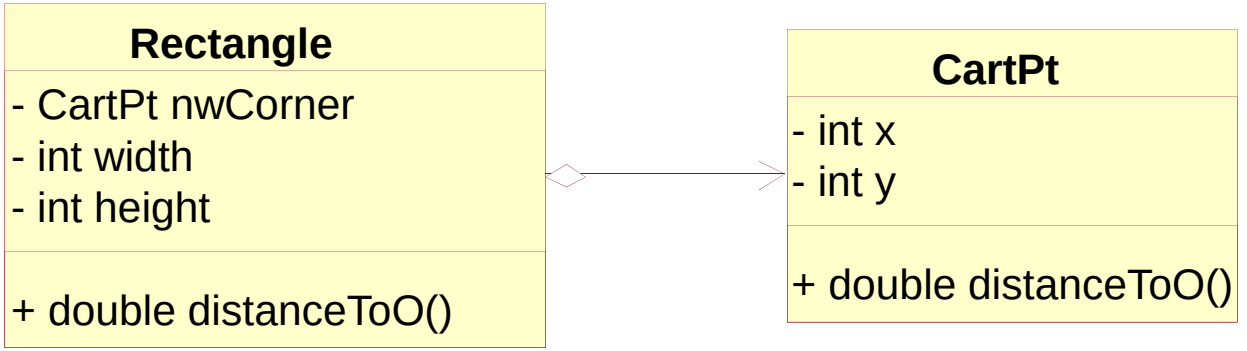
We need *two* methods:

1. Measuring the distance of a **`Rectangle`** to the origin
2. Measuring the distance of a **`CartPt`** to the origin

**Q:** Add these two methods to the class diagram

# Delegation

- Q: Which class (**`Rectangle`** or **`CartPt`**) should we put `distanceToO()` method in ?
- A: Put `distanceToO()` in both classes.
  - The **`distanceToO()`** method deals with properties of the **`CartPt`** so that we delegate this computational task to the corresponding methods in **`CartPt`** class



# `distanceToO` examples

```
CartPt p = new CartPt(3, 4);
CartPt q = new CartPt(5, 12);

Rectangle r = new Rectangle(p, 5, 17);
Rectangle s = new Rectangle(q, 10, 10);

p.distanceToO() // should produce 5
q.distanceToO() // should produce 13
r.distanceToO() // should produce 5
s.distanceToO() // should produce 13
```

# `distanceToO` purpose and signature

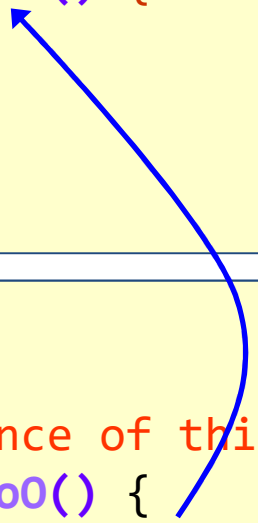
```
public class CartPt {
   private int x;
   private int y;
   public CartPt(int x, int y) { ... }

   // to compute the distance of this point to the origin
   public double distanceToO() { ... }
}
```

```
public class Rectangle {
   private CartPt nwCorner;
   private int width;
   private int height;
   public Rectangle(CartPt nwCorner, int width, int height) {
      ... }

   // to compute the distance of this Rectangle to the origin
   public double distanceToO() { ... }
}
```

# distanceToO method template

```
public class CartPt {
    ...
    // to compute the distance of this point to the origin
    public double distanceToO() {
        ...this.x...
        ...this.y...
    }
}
```

```
public class Rectangle {
    ...
    // to compute the distance of this Rectangle to the origin
    public double distanceToO() {
        ...this.nwCorner.distanceToO()...
        ...this.width...
        ...this.height...
    }
}
```



# `distanceToO` method implementation

```
public class CartPt {
   private int x;
   private int y;

   public CartPt(int x, int y) {
      this.x = x;
      this.y = y;
   }

   // to compute the distance of this CartPt to the origin
   public double distanceToO() {
      return Math.sqrt(this.x * this.x + this.y * this.y);
   }
}
```

**Tips**: Math.sqrt is the  name of the method that computes the square root of its argument as a double.

# `distanceToO` method implementation

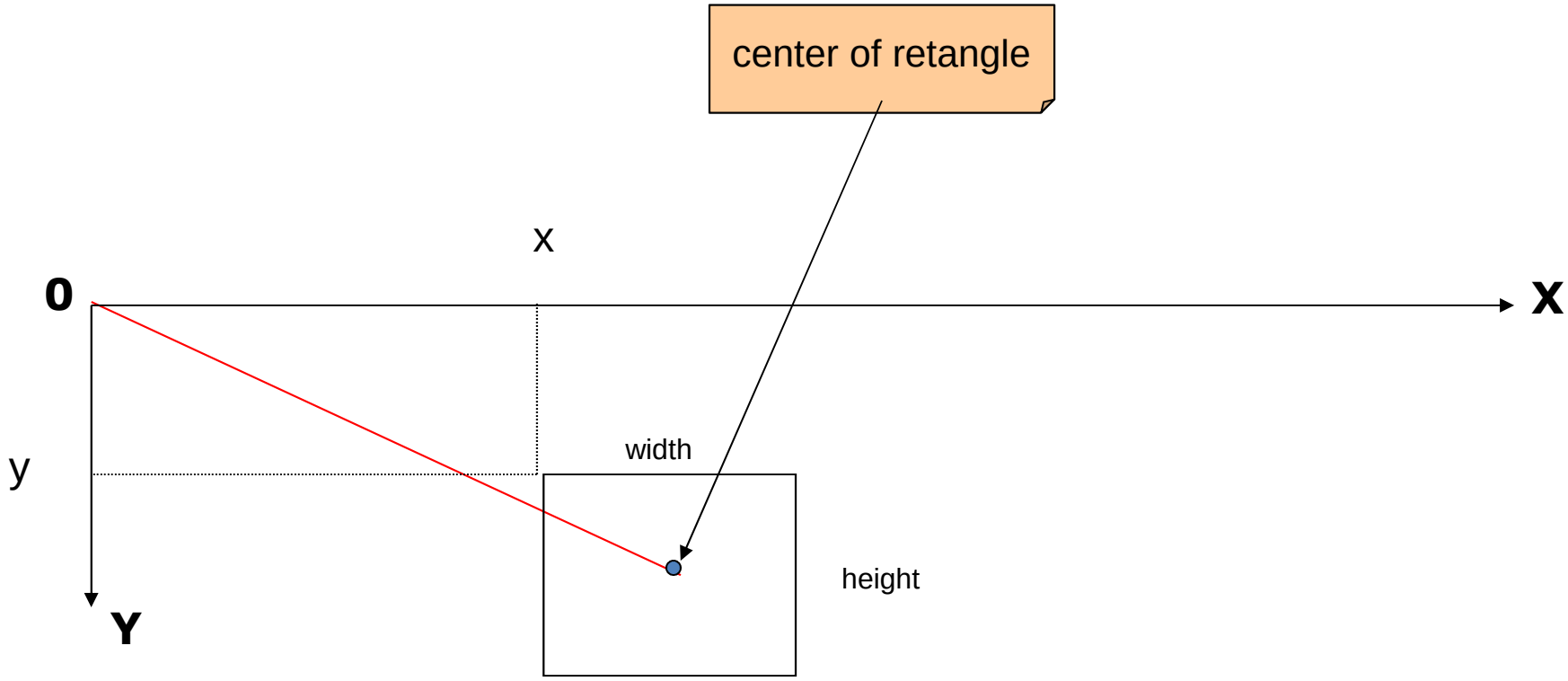
```
public class Rectangle {
   private CartPt nwCorner;
   private int width;
   private int height;

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

   // to compute the distance of this Rectangle to the origin
   public double distanceToO() {
      return this.nwCorner.distanceToO();
   }
}
```

# `distanceToO` Testing

```
public void testDistanceToO() {
   CartPt p = new CartPt(3, 4);
   Rectangle r = new Rectangle(p, 5, 17);
   assertEquals(r.distanceToO(), 5, 0.001);

   p = new CartPt(5, 12);
   r = new Rectangle(p, 10, 10);
   assertEquals(r.distanceToO(), 13, 0.001);
}
```

# Problem Extension Statement

- Compute the distance between the rectangle's center and the origin



# Solution 1

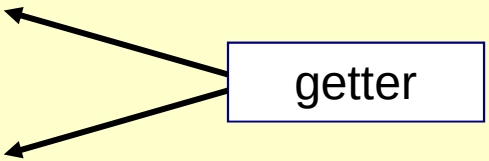
```
public class Rectangle {
   private CartPt nwCorner;
   private int width;
   private int height;
   public Rectangle(CartPt nwCorner, int width, int height) {
      this.nwCorner = nwCorner;
      this.width = width;
      this.height = height;
   }

   public double distanceToO() {
      return this.nwCorner.distanceToO();
   }

   public double distanceFromCenterToO() {
      int a = this.nwCorner.getX() + this.width/2;
      int b = this.nwCorner.getY() + this.height/2;
      return Math.sqrt(a*a + b*b);
   }
}
```

**Q**: Is it right?

**A**: right, but the delegation is not applied.

# Solution 1 (cont)

```
public class CartPt {
   private int x;
   private int y;
   public CartPt(int x, int y) {
      this.x = x;
      this.y = y;
   }

   public double distanceToO() {
      return Math.sqrt(this.x * this.x + this.y * this.y);
   }

   public int getX() {
      return this.x;
   }

   public int getY() {
      return this.y;
   }
}
```

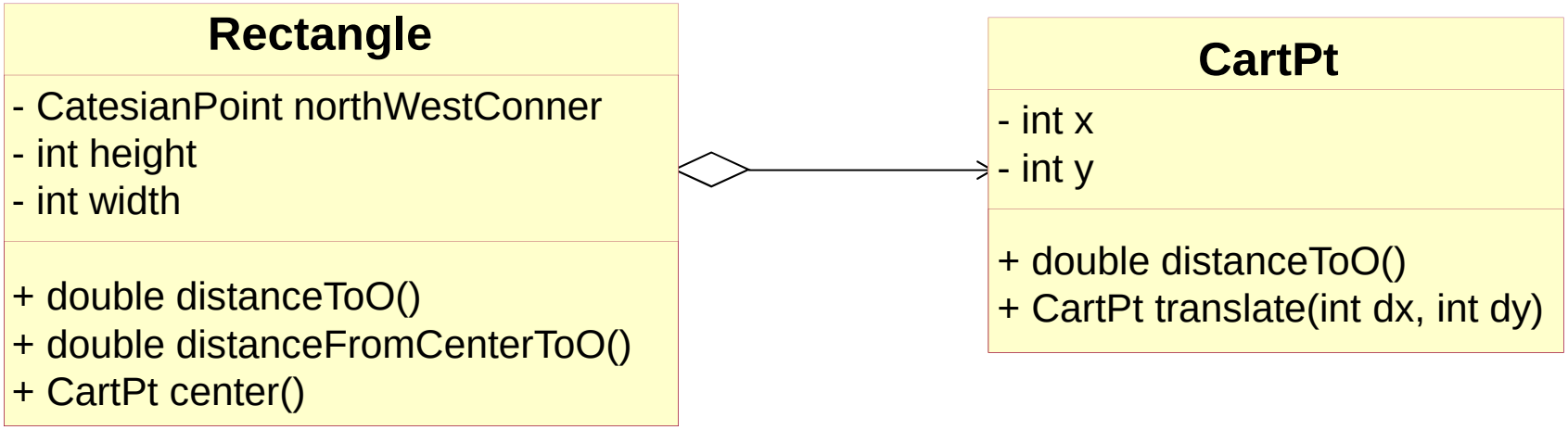


# Solution 2

```
public class Rectangle {
   private CartPt nwCorner;
   private int width;
   private int height;
   ...

   public double distanceFromCenterToO() {
      return this.center().distanceToO();
   }

   private CartPt center() {
      return this.nwCorner.translate(this.width/2, this.height/2);
   }
}
```

```
public class CartPt {
   private int x;
   private int y;
   ...
   public double distanceToO() {
      return Math.sqrt(this.x * this.x + this.y * this.y);
   }

   public CartPt translate(int dx, int dy) {
      return new CartPt(this.x + dx, this.y + dy);
   }
}
```

# Class diagram

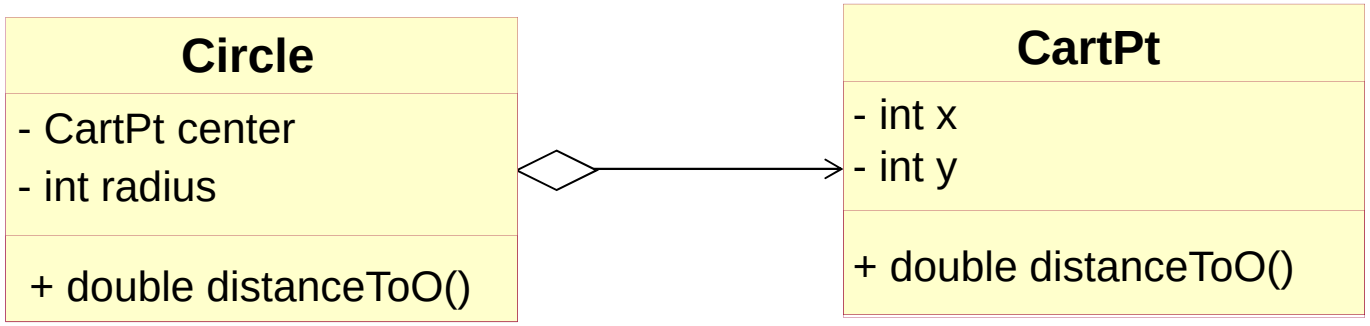


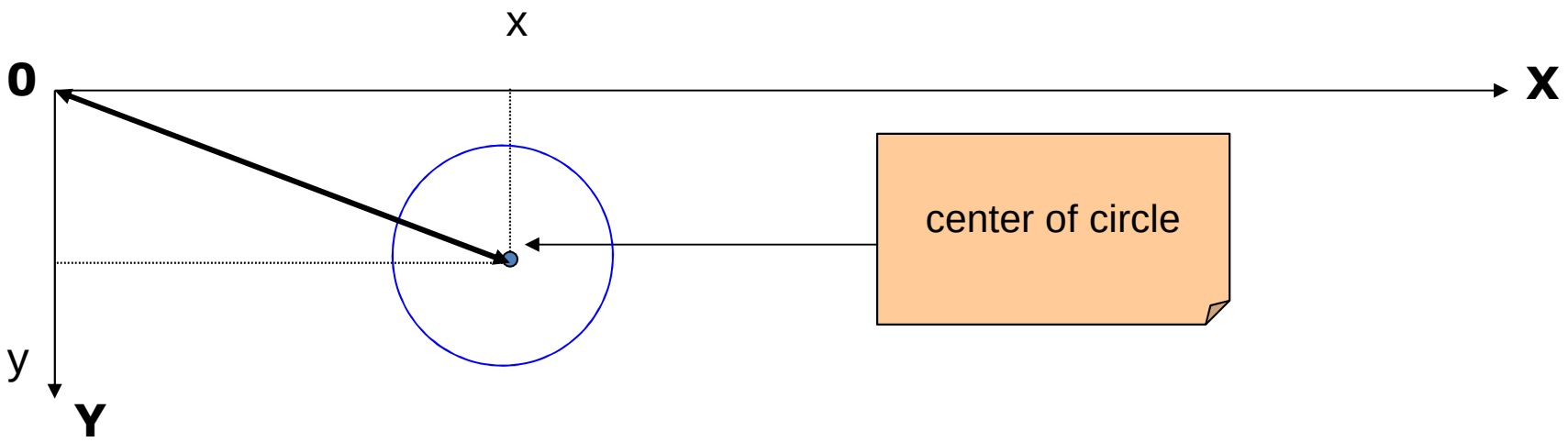
# Circle example

The circle are located on the Cartesian plane of a computer canvas, which has its center and radius.

1. Compute the distance form circle to the origin
2. Computing the perimeter of a circle
3. Computing the area of a circle.
4. Computes the area of a ring, that is, this disk with a hole in the center

# Distance form circle to the origin



# distanceToO template

```
public class Circle {
   private CartPt center;
   private int radius;

   public Circle(CartPt center, int radius) {
      this.center = center;
      this.radius = radius;
   }

   public double area() {
      return Math.PI*this.radius*this.radius;
   }

  // to compute the distance of this Circle to the origin
  public doublle distanceToO() {
      ...this.center.distanceToO()...
      ...this.radius...
}
```

# distanceToO body

```
public class Circle {
   private CartPt center;
   private int radius;

   public Circle(CartPt center, int radius) {
      this.center = center;
      this.radius = radius;
   }

   // to compute the distance of this Circle to the origin
   public double distanceToO() {
      return this.center.distanceToO();
   }
}
```

# distanceToO test

```
public void testdistanceToO() {
    Circle c1 = new Circle(new CartPt(3, 4), 5);
    Circle c2 = new Circle(new CartPt(5, 12), 10);
    Circle c3 = new Circle(new CartPt(6, 8), 20);
    assertEquals(c1.distanceToO(), 5.0, 0.001);
    assertEquals(c2.distanceToO(), 13.0, 0.001);
    assertEquals(c3.distanceToO(), 10.0, 0.001);
}
```

# `perimeter` template

```
public class Circle {
   private CartPt center;
   private int radius;

   public Circle(CartPt center, int radius) {
      this.center = center;
      this.radius = radius;
   }

   // Compute the perimeter of the circle
   public double perimeter () {
     ...this.distanceToO()...
     ...this.center.distanceToO()
     ...this.radius...
   }
}
```

# perimeter body

```
public class Circle {
   private CartPt center;
   private int radius;

   public Circle(CartPt center, int radius) {
      this.center = center;
      this.radius = radius;
   }

   //Compute the perimeter of the circle
   public double perimeter() {
     return 2* Math.PI * this.radius;
   }
}
```

# perimeter Test

```
public void testPerimeter() {
    Circle c1 = new Circle(new CartPt(3, 4), 5);
    Circle c2 = new Circle(new CartPt(5, 12), 10);
    Circle c3 = new Circle(new CartPt(6, 8), 20);
    assertEquals(c1.perimeter(), 31.42, 0.001);
    assertEquals(c2.perimeter(), 62.83, 0.001);
    assertEquals(c3.perimeter(), 125.66, 0.001);
}
```

# area template

```
public class Circle {
   private CartPt center;
   private int radius;

   public Circle(CartPt center, int radius) {
      this.center = center;
      this.radius = radius;
   }
   ...

   //Compute the area of the circle
   public double area () {
     ...this.distanceToO()...
     ...this.perimeter()...
     ...this.center.distanceToO()...
     ...this.radius...
   }
}
```

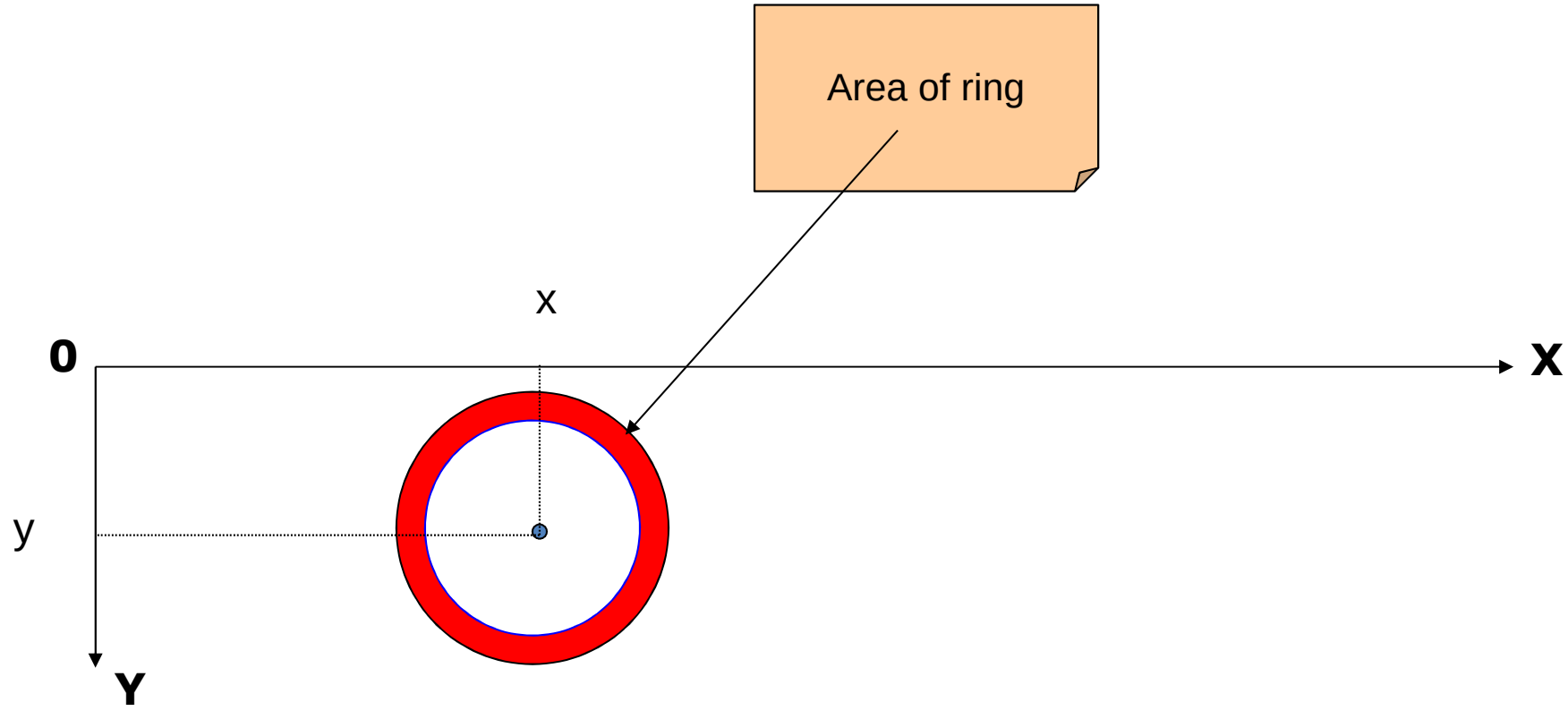
# area body

```
public class Circle {
   private CartPt center;
   private int radius;

   public Circle(CartPt center, int radius) {
      this.center = center;
      this.radius = radius;
   }

   //Compute the perimeter of the circle
   public double area() {
      return Math.PI * this.radius * this.radius;
   }
}
```

# area Test

```
public void testArea() {
    Circle c1 = new Circle(new CartPt(3, 4), 5);
    Circle c2 = new Circle(new CartPt(5, 12), 10);
    Circle c3 = new Circle(new CartPt(6, 8), 20);

    assertEquals(c1.area(), 78.54, 0.001);
    assertEquals(c2.area(), 314.16, 0.001);
    assertEquals(c3.area(), 1256.64, 0.001);
}
```

# Area of ring



# `area` template

```
public class Circle {
   private CartPt center;
   private int radius;
   public Circle(CartPt center, int radius) {
      this.center = center;
      this.radius = radius;
   }
   // Compute the area of the circle
   public double area() {
      return Math.PI * this.radius * this.radius;
   }

   // Compute the area of the ring
   public double area(Circle that) {
      ...this.center...this.radius...
      ...this.distanceToO()...this.perimeter()...this.area()...
      ...that.center...that.radius...
      ...that.distanceToO()...that.perimeter()...that.area()...
  }
}
```

# area body

```
public class Circle {
   private CartPt center;
   private int radius;
   public Circle(CartPt center, int radius) {
      this.center = center;
      this.radius = radius;
   }
   // Compute the area of the circle
   public double area() {
      return Math.PI * this.radius * this.radius;
   }

   // Compute the area of the ring
   public double area(Circle that) {
      return Math.abs(this.area() - that.area());
   }
}
```

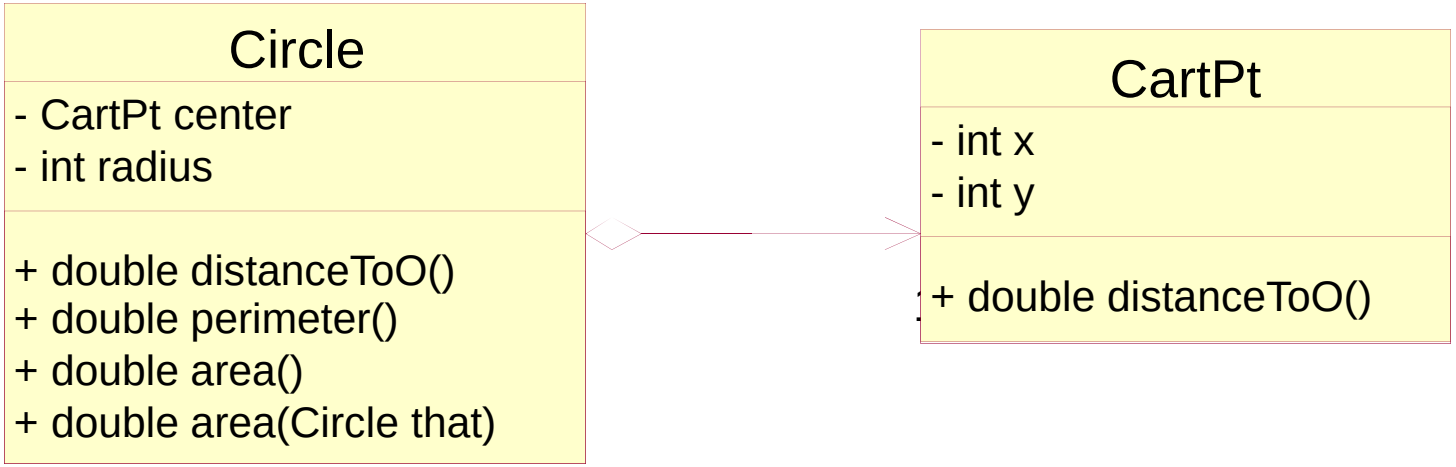
# area Test

```
public void testArea() {
    Circle c1 = new Circle(new CartPt(3, 4), 5);
    Circle c2 = new Circle(new CartPt(5, 12), 10);
    Circle c3 = new Circle(new CartPt(6, 8), 20);

    assertEquals(c1.area(), 78.54, 0.01);
    assertEquals(c2.area(), 314.16, 0.01);
    assertEquals(c3.area(), 1256.64, 0.01);

    assertEquals(c2.area(c1), 235.62, 0.01);
    assertEquals(c3.area(c1), 1178.1, 0.01);
    assertEquals(c3.area(c2), 942.48, 0.01);
}
```

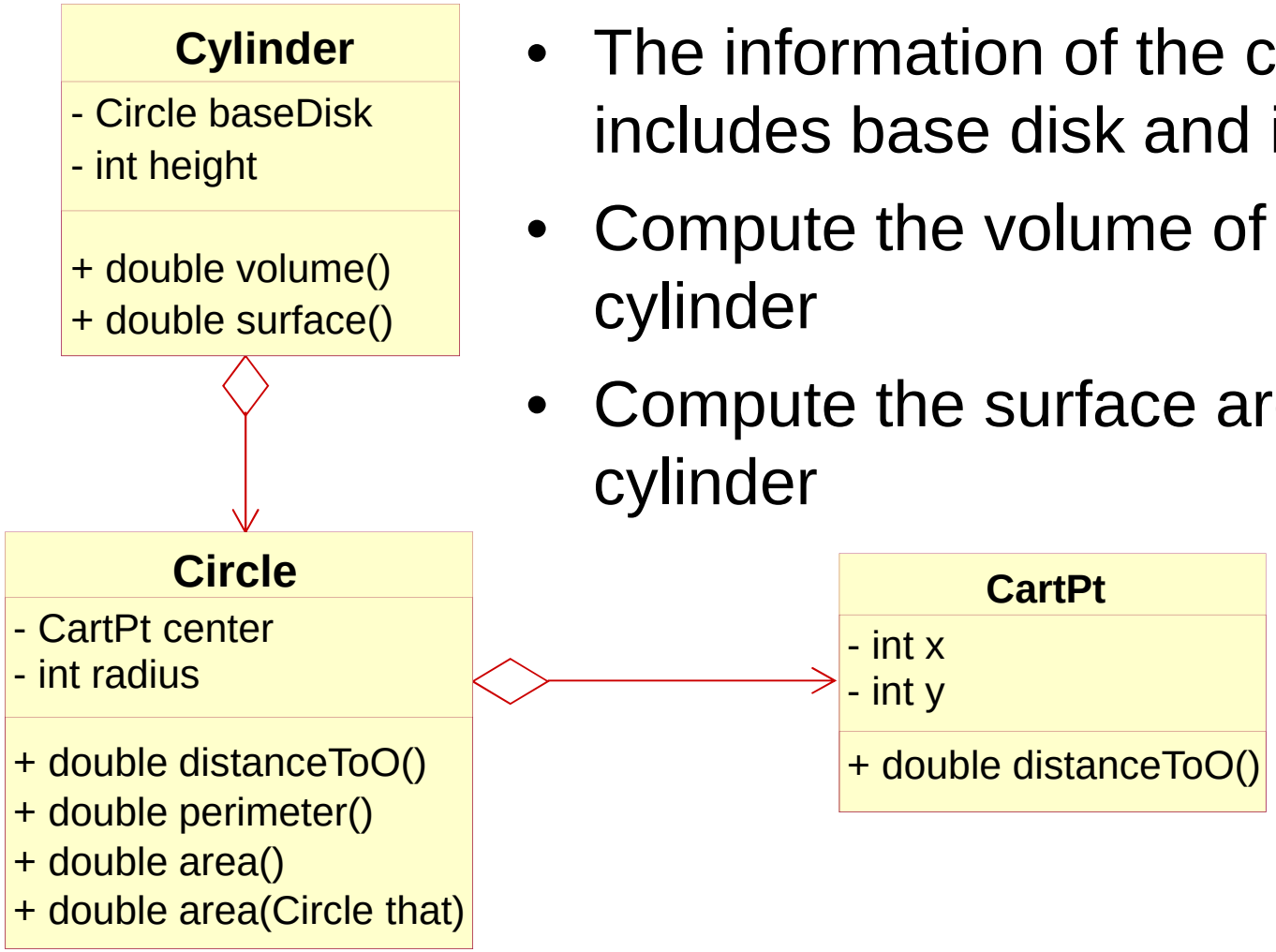
# Overloading method

- **Q**: what happen with the same name `area()` and `area(Circle)` method?
- **A:**
  - Method `area()` and `area(Cirlce )` in class `Cirlce` have the same name but different parameter is called overloading.
  - When we invoke overloading methods, the method with appropriate argument will do

# Class diagram



# Cylinder example




- The information of the cylinder includes base disk and its height.
- Compute the volume of the cylinder
- Compute the surface area of the cylinder

# `volume` method template

```
public class Cylinder {
   private Circle baseDisk;
   private int height;

   public Cylinder(Circle baseDisk, int height) {
      this.baseDisk = baseDisk;
      this.height = height;
   }

   // Compute the volume of the cylinder
   public double volume() {
      ...this.baseDisk.distanceToO()
      ...this.baseDisk.perimeter()...this.baseDisk.area()...
      ...this.height...
   }
}
```

# volume method body

```
public class Cylinder {
   private Circle baseDisk;
   private int height;

   public Cylinder(Circle baseDisk, int height) {
      this.baseDisk = baseDisk;
      this.height = height;
   }

   // Compute the volume of the cylinder
   public double volume() {
      return this.baseDisk.area() * this.height;
   }
}
```

# `volume` method test

```
public void testVolume(){
   Circle c1 = new Circle(new CartPt(3,4), 5);
   Circle c2 = new Circle(new CartPt(5,12), 10);
   Circle c3 = new Circle(new CartPt(6,8), 20);

   Cylinder cy1 = new Cylinder(c1, 10);
   Cylinder cy2 = new Cylinder(c2, 30);
   Cylinder cy3 = new Cylinder(c3, 40);

   assertEquals(cy1.volume(), 785.4, 0.001);
   assertEquals(cy2.volume(), 9424.77, 0.001);
   assertEquals(cy3.volume(), 50265.48, 0.001);
}
```

# `surface` method template

```
public class Cylinder {
   private Circle baseDisk;
   private int height;
   public Cylinder(Circle baseDisk, int height) {
      this.baseDisk = baseDisk;
      this.height = height;
   }

   // Compute the surface of the cylinder
   public double surface(){
      ...this.baseDisk.distanceToO()
      ...this.baseDisk.perimeter()...this.baseDisk.area()...
      ...this.height...
   }
}
```

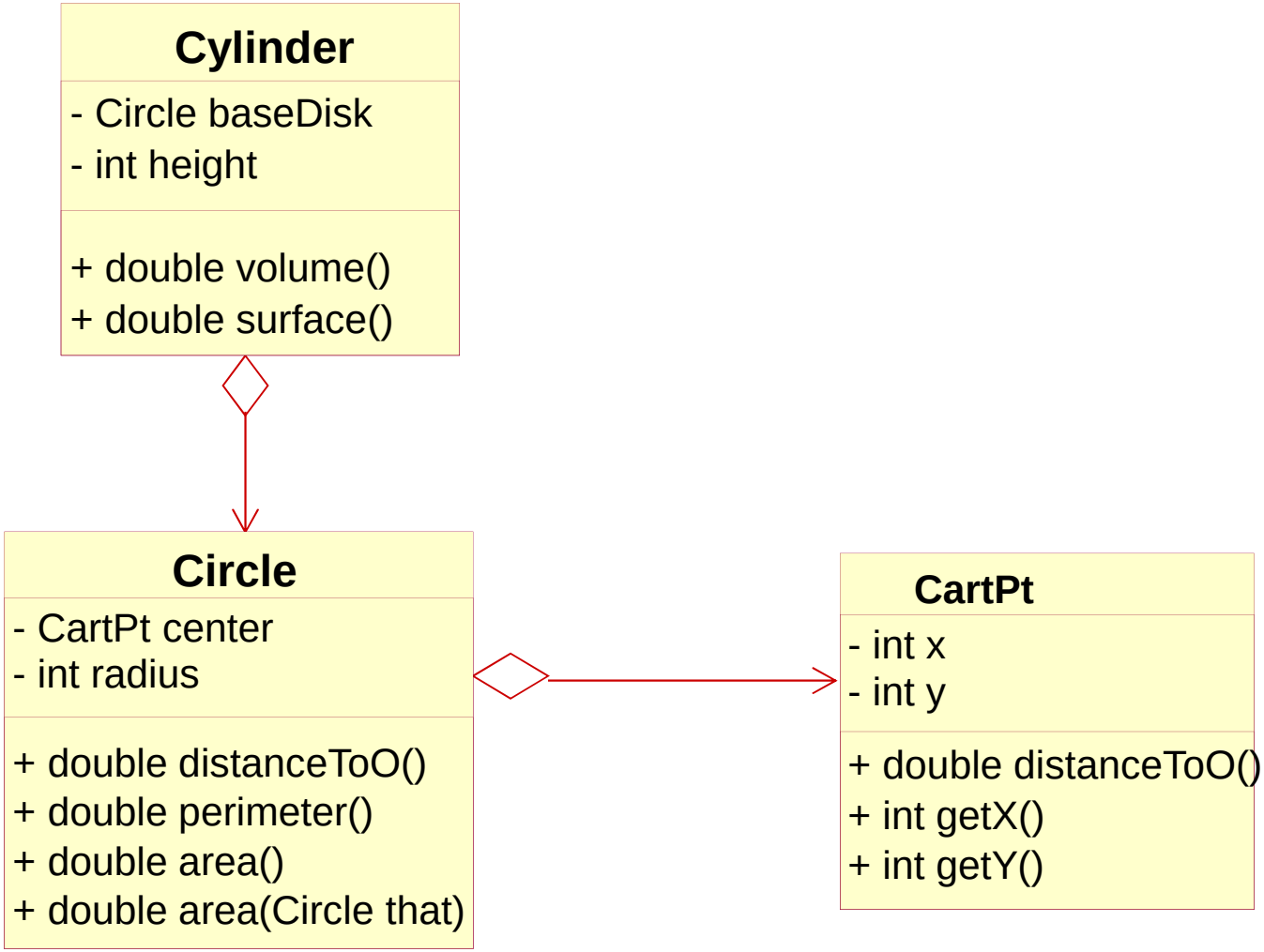
# surface method body

```
public class Cylinder {
   private Circle baseDisk;
   private int height;
   public Cylinder(Circle baseDisk, int height) {
      this.baseDisk = baseDisk;
      this.height = height;
   }

   // Compute the volume of the cylinder
   public double volume() {
      return this.baseDisk.area() * this.height;
   }

   // Compute the surface of the cylinder
   public double surface() {
      return this.baseDisk.perimeter() * this.height;
   }
}
```

# `surface` method test

```
public void testSurface() {
   Circle c1 = new Circle(new CartPt(3, 4), 5);
   Circle c2 = new Circle(new CartPt(5, 12), 10);
   Circle c3 = new Circle(new CartPt(6, 8), 20);

   Cylinder cy1 = new Cylinder(c1, 10);
   Cylinder cy2 = new Cylinder(c2, 30);
   Cylinder cy3 = new Cylinder(c3, 40);

   assertEquals(cy1.surface(), 314.16, 0.001);
   assertEquals(cy2.surface(), 1884.95, 0.001);
   assertEquals(cy3.surface(), 5026.54, 0.01);
}
```

# Class diagram



# Excercises

# Exercise 3.1

Develop a "real estate assistant" program. The "assistant" helps the real estate agent locate houses of interest for clients. The information about a house includes its kind, the number of rooms, the asking price, and its address. An address consists of a house number, a street name, and a city.

- Represent the following examples using your classes:
  - Ranch, 7 rooms, $375,000, 23 Maple Street, Brookline
  - Colonial, 9 rooms, $450,000, 5 Joye Road, Newton
  - Cape, 6 rooms, $235,000, 83 Winslow Road, Waltham
- Note:
  - Ranch: A ranch is a large farm used for raising animals, especially cattle, horses or sheep.
  - Colonial: A colonial building or piece of furniture was built or made in a style that was popular in American in the 17$^{th}$ and 18$^{th}$ centuries.
  - Cape: A cape is a large piece of land that sticks out into the sea from the coast.

# Exercise 3.1

Develop the following methods for the class House:

1. `hasMoreRooms`, which determines whether one house has more rooms than some other house;
2. `inThisCity`, which checks whether the advertised house is in some given city (assume we give the method a city name);
3. `sameCity`, which determines whether one house is in the same city as some other house.

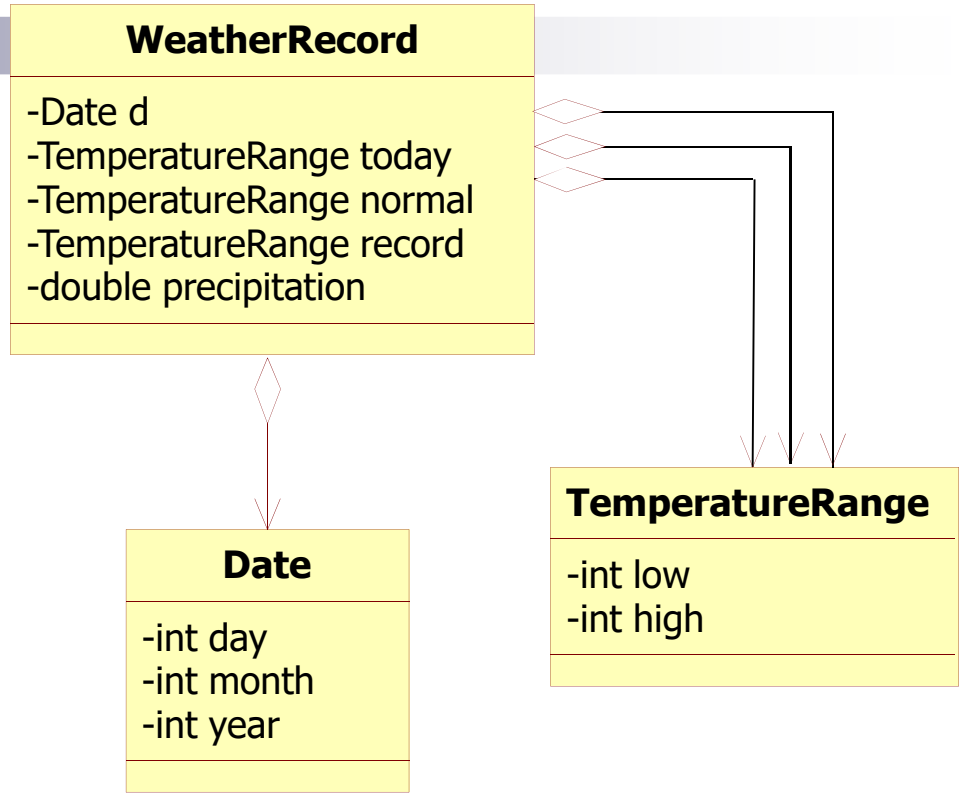
# Exercise 3.2

. . . Develop a program that assists bookstore employees. For each book, the program should track the book's title, its price, its year of publication, and the author. A author has a name and birth year.

- Develop the following methods for this class:
  - `currentBook` that checks whether the book was published in 2004 or 2003;
  - `currentAuthor` that determines whether a book was written by a current author (born after 1940);
  - `thisAuthor` that determines whether a book was written by the specified author;
  - `sameAuthor` that determines whether one book was written by the same author as some other book;
  - `sameGeneration` that determines whether two books were written by two authors born less than 10 year apart.

# Exercise 3.3

- Provides the data definition for a weather recording program.




- Develop the following methods:
  - **`withinRange`**, which determines whether today's high and low were within the normal range;
  - **`rainyDay`**, which determines whether the precipitation is higher than some given value;
  - **`recordDay`**, which determines whether the temperature broke either the high or the low record.

# Exercises 3.4 (Lab hours)

- Develop a program that can assist railway travelers with the arrangement of train trips.
- The available information about a specific train includes its schedule, its route, and whether it is local.
- The route information consists of the origin and the destination station.
- A schedule specifies the departure and the arrival (clock) times when the train leaves and when it arrives.
- ClockTime consists of the hour (of the day) and the minutes (of the hour).
- The customer want to know:
  - Does his destination station match the destination of the train trip?
  - What time does the train start ?
  - How long does the train trip take?

# Class Diagram