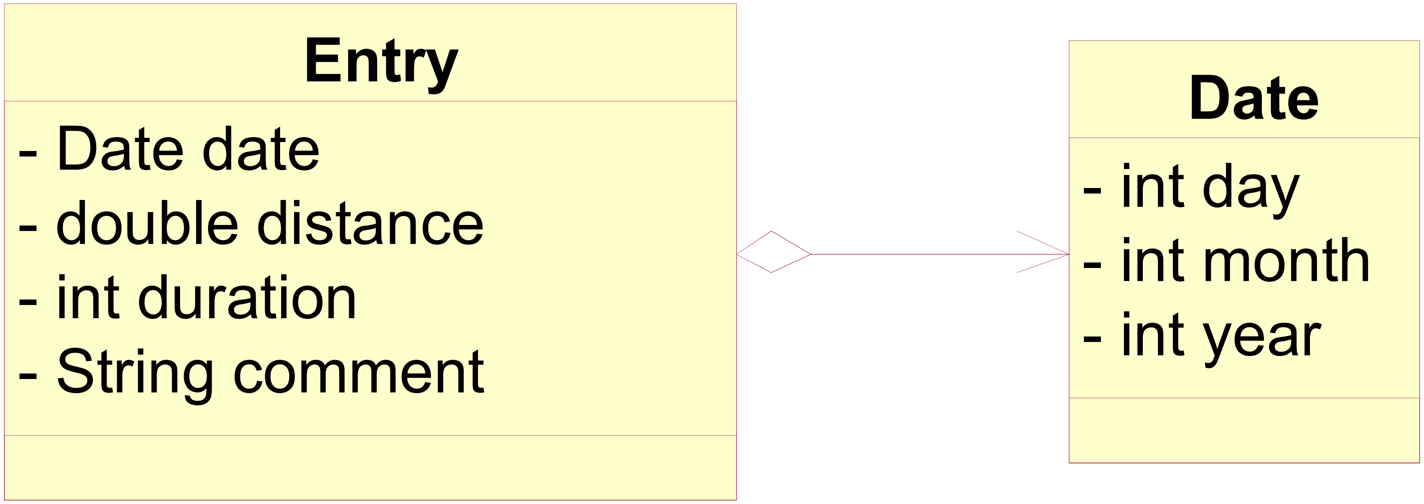
Class References, Object Containment and Methods

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;
   }
}
```

reference

```
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

object containment

```
import junit.framework.*;
public class EntryTest extends TestCase {

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

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

      Date date1 = new Date(23, 6, 2004);
      new Entry(date1, 26.2, 159, "exhausted");
   }
}
```

The public or private modifiers for attribute and method

- **None modifier**: Classes in the same package can access this attribute / method.

- **`public`**: Classes in all packages can access this attribute / method.

- **`private`**: Only the class itself can access this attribute / method.

⇨ Encapsulation

Encapsulation

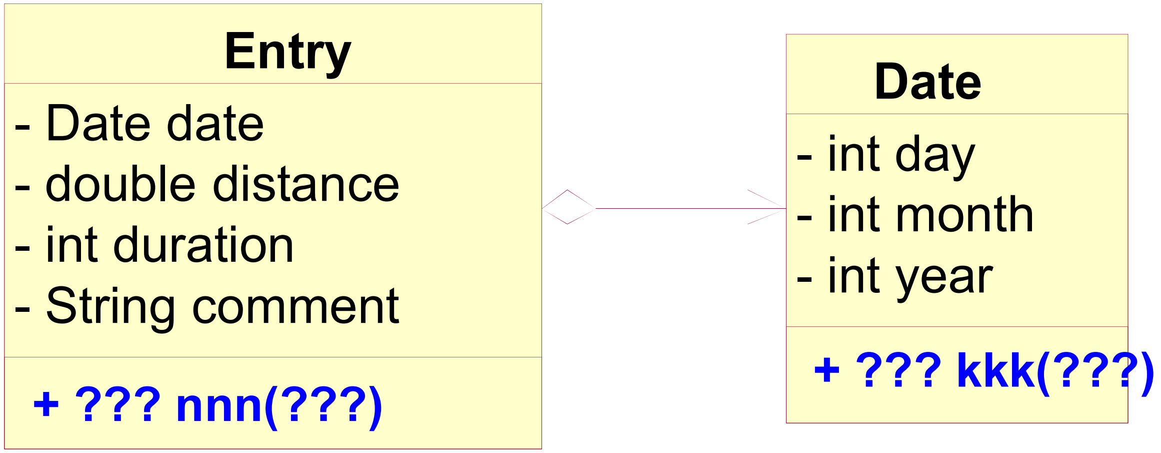
- A mechanism used to hide the data, internal structure, and implementation details of an object. All interaction with the object is through a public interface of operations.
  - Data inside the object is only accessible by the object's operations. No other object can reach inside the object and change its attribute values.
- The reason for hiding features is to:

  (1) keep users from touching parts of the object they shouldn't touch;

  (2) allows creator of the object to change the object's internal working without affecting the users of the object.
- Apply encapsulation for class:
  - The data fields are ***private***
  - The allowed methods are ***public***

Methods for containment

Add methods to the `Entry`



Entry

- Date date
- double distance
- int duration
- String comment

+ ??? nnn(???)

Date

- int day
- int month
- int year

+ ??? kkk(???)

Java template for Entry

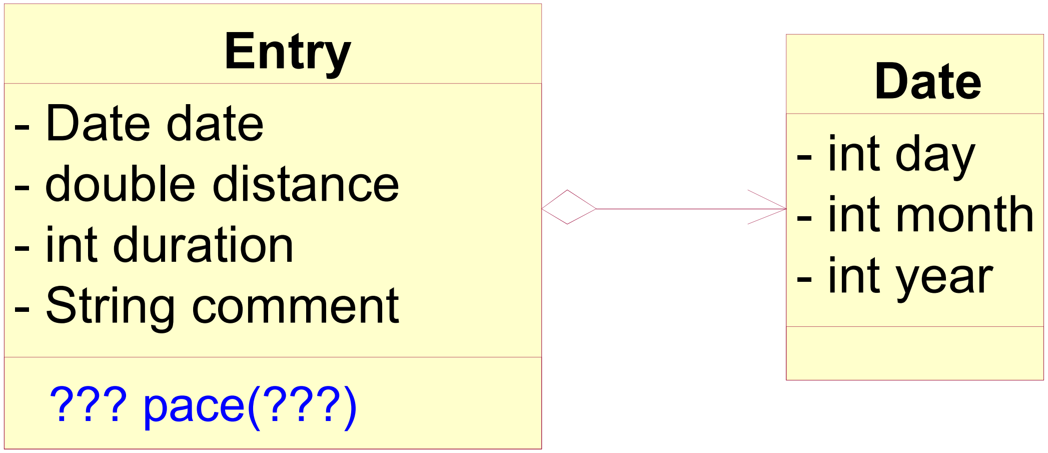
```
public class Entry {
   private Date date;
   private double distance;
   private int duration;
   private String comment;
   public Entry(Date date, double distance, int duration,
         String comment) {
      this.date = date;
      this.distance = distance;
      this.duration = duration;
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

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

Design pace() method (con't)

- Unit testing

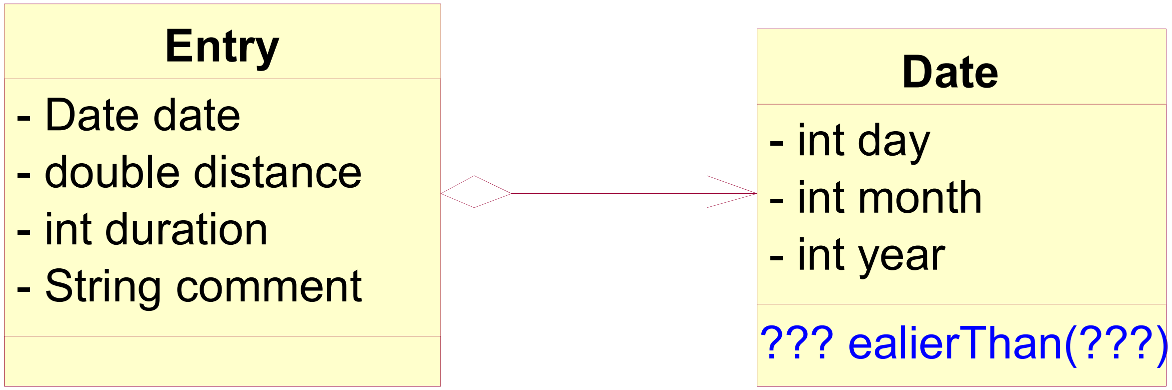
```
pubblic class EntryTest extends TestCase {
   ...

   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

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;
}
```

Design earlyThan() method

Implement

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

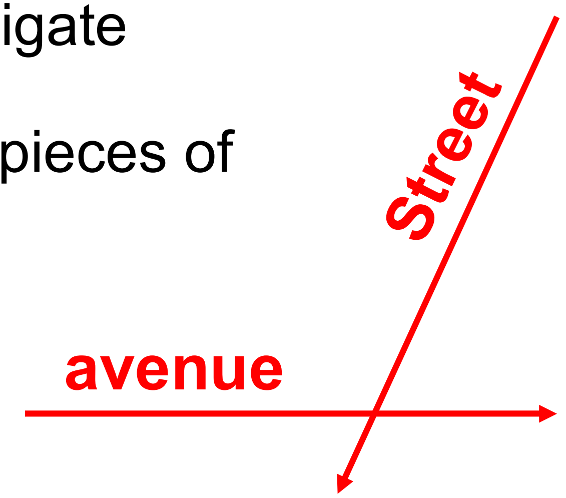
```
public boolean earlierThan(Date that) {
   if (this.year < that.year) {
      return true;
   }
   else {
      if (this.year > that.year) {
         return false;
      }
      else {
         if (this.month < that.month) {
             return true;
         }
         else {
            if (this.month > that.month) {
               return false;
            }
            else { if (this.day < that.day) return true; }
                   else 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, 2003);
      Date date4 = new Date(15, 12, 2004);
      Date date5 = new Date(31, 12, 2004);

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

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

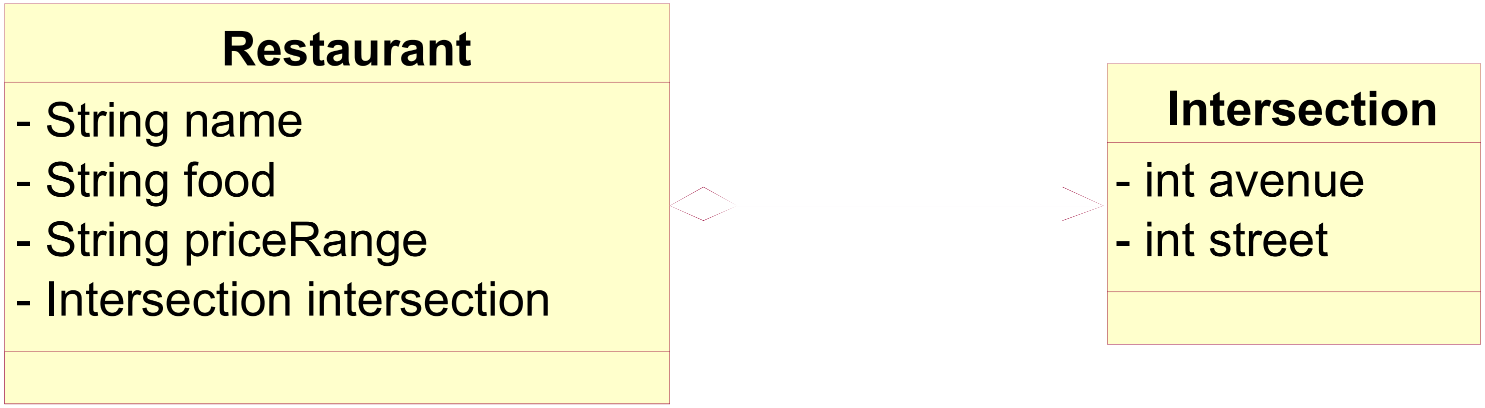
Restaurant example

- 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;

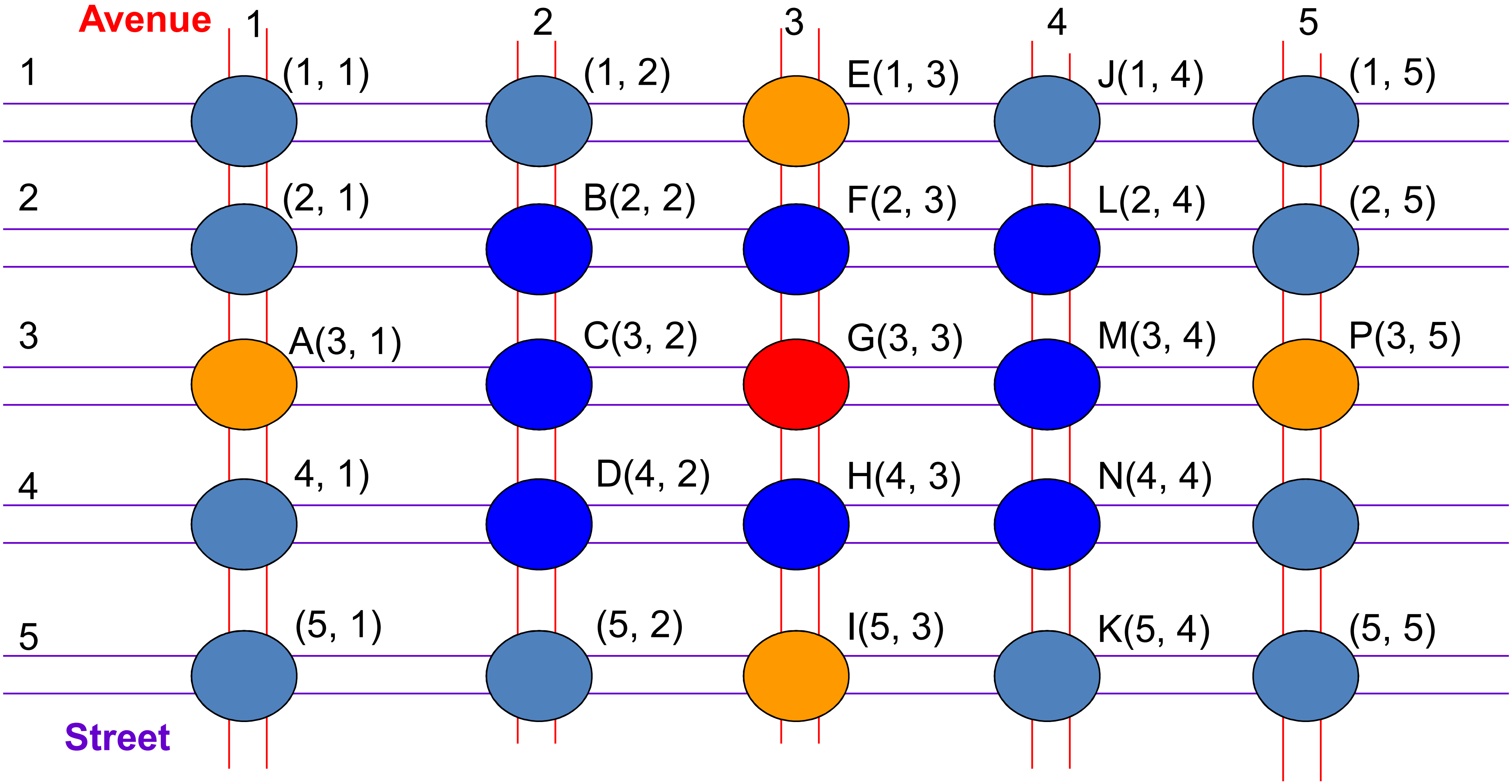
Class Diagram



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

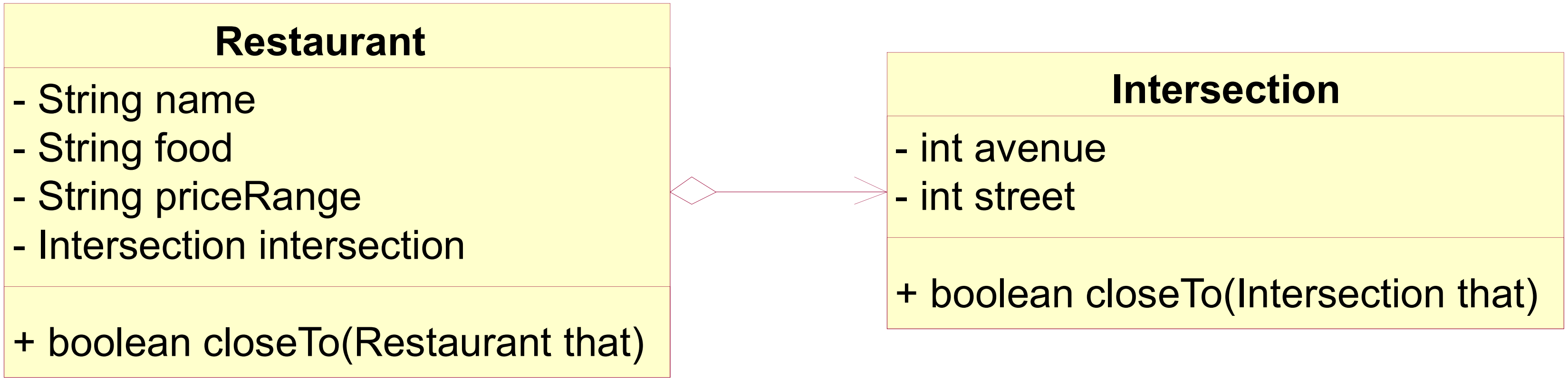
🔵 🟠 Intersections not “close” to the considered Intersection

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



Restaurant

- String name
- String food
- String priceRange
- Intersection intersection

+ boolean closeTo(Restaurant that)

Intersection

- int avenue
- int street

+ boolean closeTo(Intersection that)

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
```

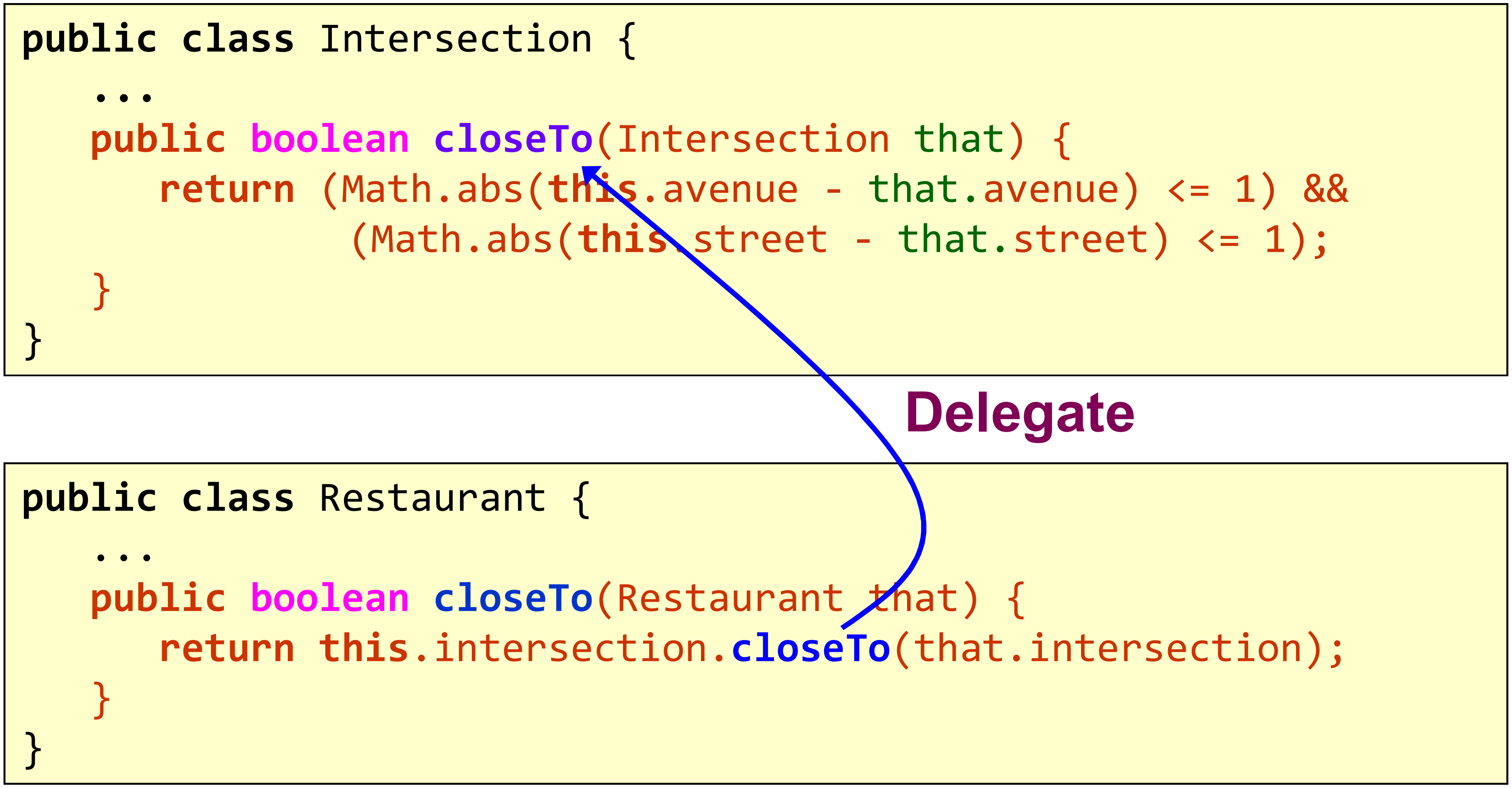
closeTo template in Intersection class

```
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.closeTo(...)...
      ...that.name... that.food...
      ...that.priceRange...
      ...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);
   }
}
```

Delegate

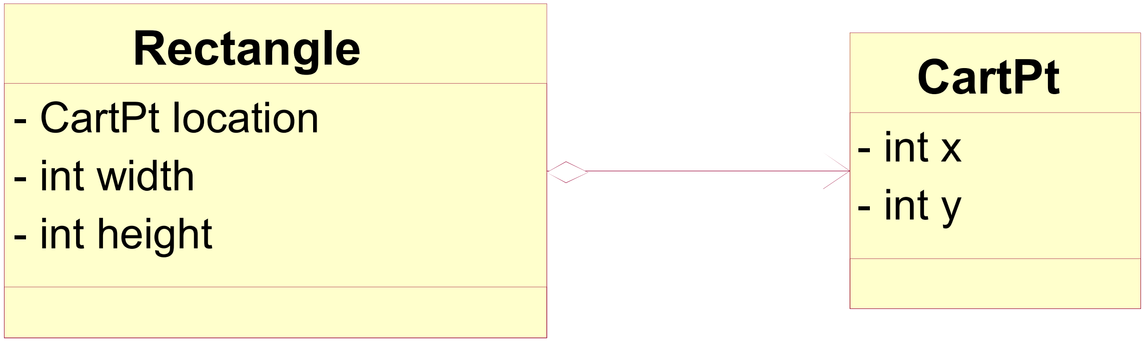
```
public class Restaurant {
   ...
   public boolean closeTo(Restaurant that) {
      return this.intersection.closeTo(that.intersection);
   }
}
```

Unit Testing

```
pubblic class RestaurantTest extends TestCase {
   public void testCloseTo() {
      Intersection i1 = new Intersection(3, 3);
      Intersection i2 = new Intersection(3, 2);
      assertTrue(i1.closeTo(i2));
      assertFalse(i1.closeTo(new Intersection(3, 5));
      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.



Rectangle

- CartPt location
- int width
- int height

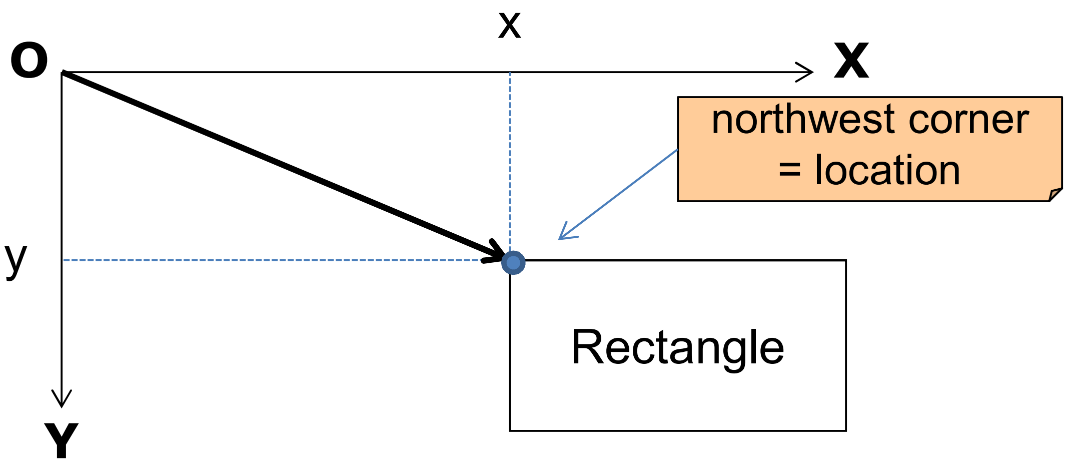
CartPt

- int x
- int y

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

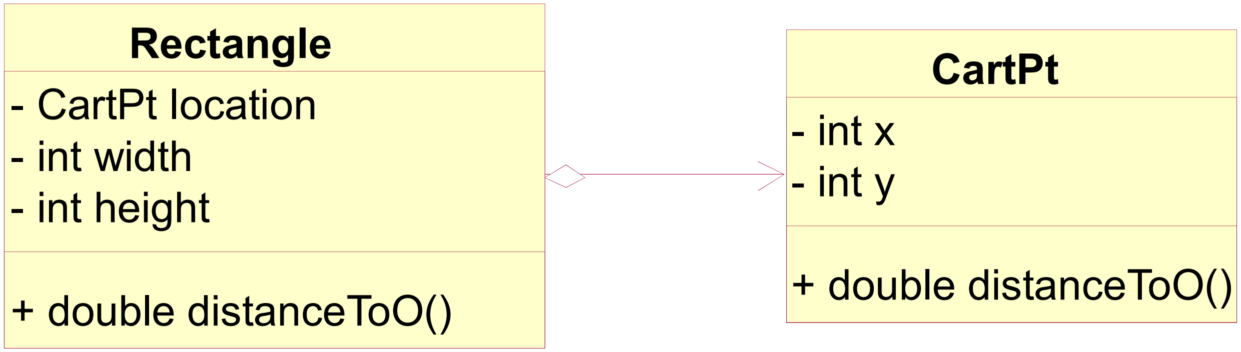
We need *two* methods:

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 location;
   private int width;
   private int height;
   public Rectangle(CartPt location, 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.location.distanceToO()...
      ...this.width...
      ...this.height...
   }
}
```

distanceToO method implementation

```
public class CartPt {
   private int x;
   private int 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);
   }
}
```

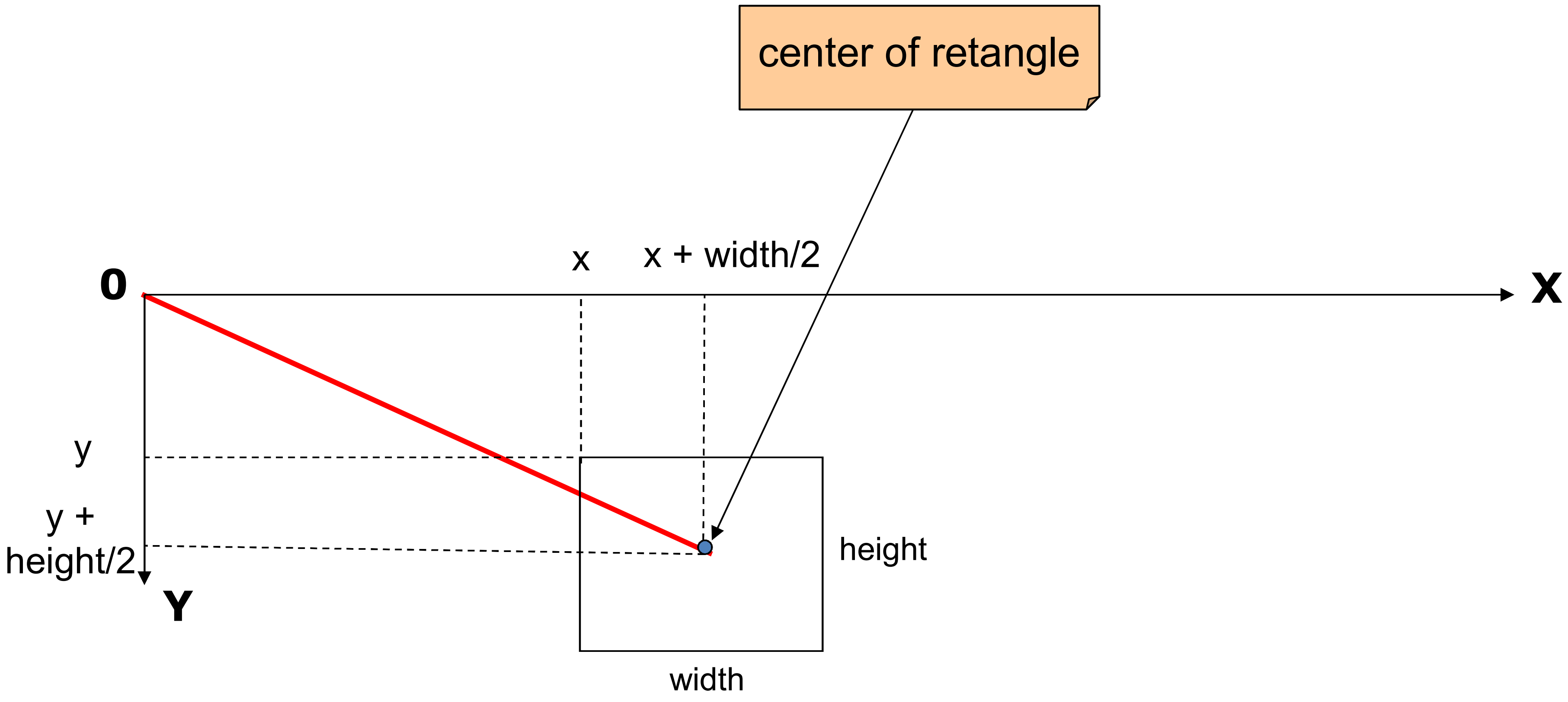
```
public class Rectangle {
   private CartPt location;
   private int width; private int height;
   ...
   // to compute the distance of this Rectangle to the origin
   public double distanceToO() {
      return this.location.distanceToO();
   }
}
```

distanceToO Testing

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

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

Problem Extension Statement

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



Solution 1: Don't delegate – Bad!

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

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

   public double distanceFromCenterToO() {
      int xc = this.location.getX() + this.width/2;
      int yc = this.location.getY() + this.height/2;
      return Math.sqrt(xc * xc + yc * yc);
      // delegate distanceTYoO to the CartPt
      // return new CartPt(xc, yc).distanceToO();
   }
}
```

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 distanceTo0() {
      return Math.sqrt(this.x * this.x + this.y * this.y);
   }

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

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

Getters

Solution 2: Using delegation

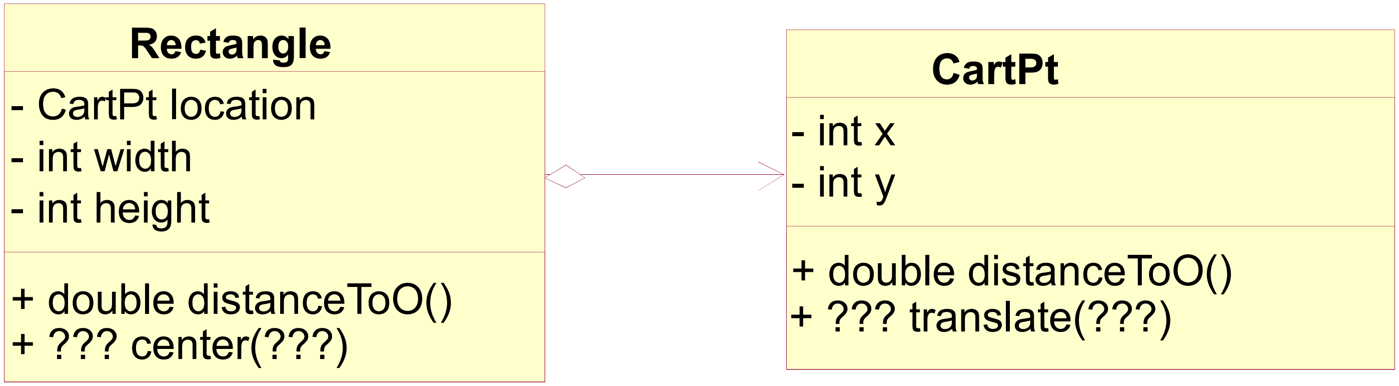
To compute the distance between the rectangle’s center and the origin:

- First, specify the `center point` of `Rectangle` Delegate the computing to `location` translate (width/2, height/2) offset
- Then delegate the compute **`distancetoO()`** to the center point

Specify center point of `Rectangle`

Specify center point of `Rectangle`

- Delegate the computing to `location` translate (width/2, height/2) offset



Method signature

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

   // determine the center of this point
   public CartPt center() { }
}
```

```
public class CartPt {
   private int x;
   private int y;
   ...
   // this point translate to (dx, dy) offset
   public CartPt translate(int dx, int dy) { … }
}
```

Examples

```
new CartPt(3, 4).translate(2, 8)
should be new CartPt(5, 12)
new CartPt(1, 4).translate(3, 5)
should be new CartPt(4, 9)

new Rectangle(new CartPt(3, 4), 4, 17).center()
should be new CartPt(5, 12)
new Rectangle(new CartPt(1, 4), 7, 10).center()
should be new CartPt(4, 9)
```

translate() method in CartPT

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

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

   public boolean equals(Object obj) {
      if (null == obj || !(obj instanceof CartPt))
         return false;
      else {
         CartPt that = (CartPt) obj;
         return (this.x == that.x)
             && (this.y == that.y);
      }
   }
}
```

Implement equals() method for test

translate() method test

```
public class RectangleTest extends TestCase {
   ...
   public void testCartPtTranslate() {
      assertEquals(new CartPt(3, 4).translate(2, 8),
                   new CartPt(5, 12));
      assertEquals(new CartPt(1, 4).translate(3, 5),
                   new CartPt(4, 9));
      CartPt q = new CartPt(4, 7);
      assertEquals(q.translate(0, 0), new CartPt(4, 7));
   }
}
```

Specify center point of Rectangle

Delegate the computing to location translate (width/2, height/2) offset

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

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

Q: How to find the value of the center?

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

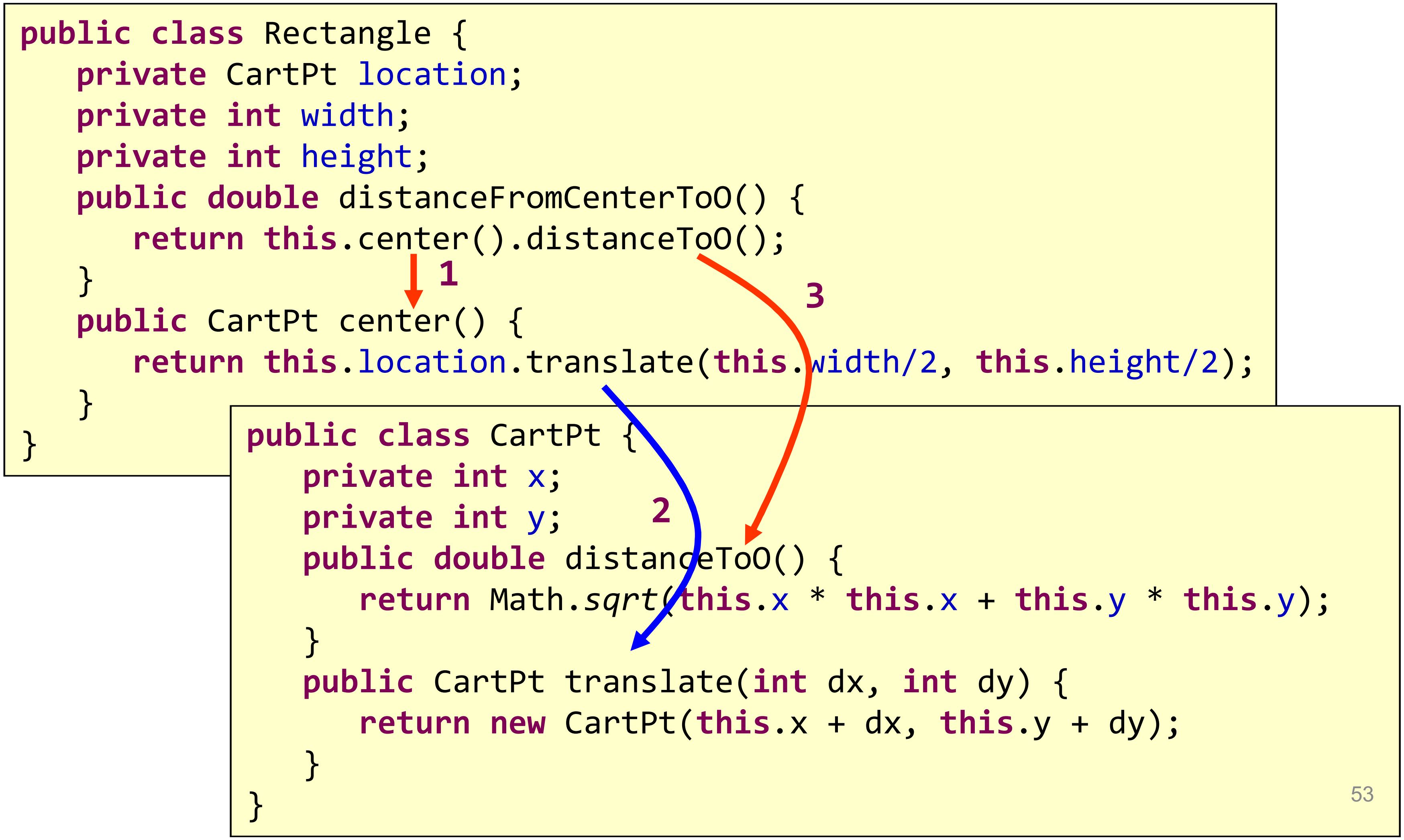
Delegate

point translate to (dx, dy) offset →

center() method test

```
public class RectangleTest extends TestCase {
   ...
   public void testCenter() {
      assertEquals(new CartPt(3, 4).translate(2, 8),
                   new CartPt(5, 12));
      assertEquals(new CartPt(1, 4).translate(3, 5),
                   new CartPt(4, 9));

      assertEquals(new Rectangle(new CartPt(3, 4), 4, 17)
        .center(), new CartPt(5, 12));
      Rectangle re = new Rectangle(new CartPt(1, 4), 7, 10);
      assertEquals(re.center(), new CartPt(4, 9));
   }
}
```

Compute distancetoO() of the center point



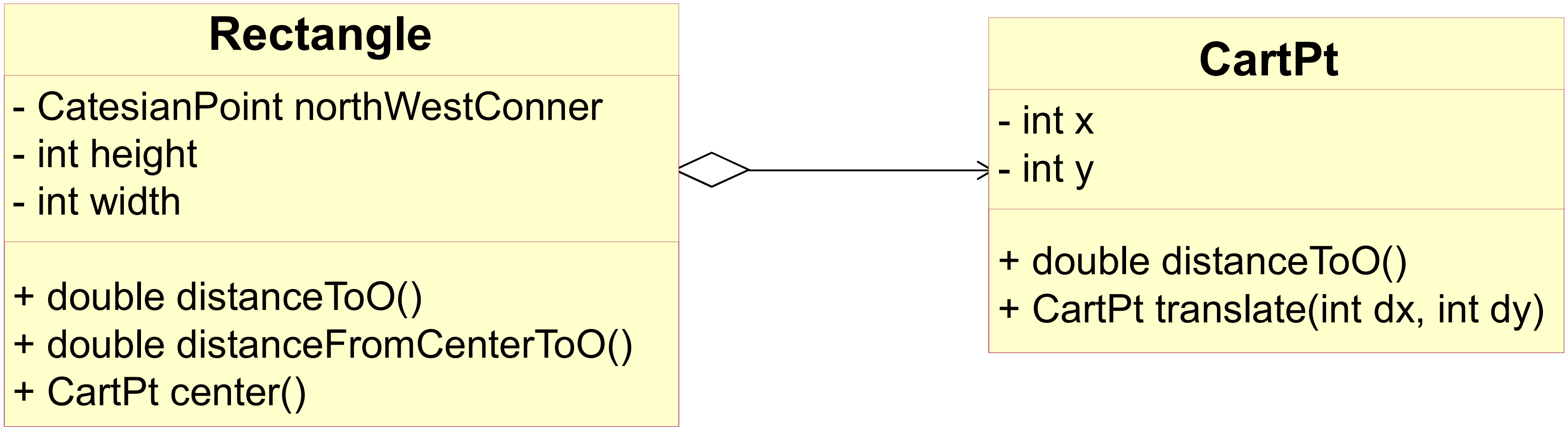
```
public class Rectangle {
   private CartPt location;
   private int width;
   private int height;
   public double distanceFromCenterToO() {
      return this.center().distanceToO();
   }
   public CartPt center() {
      return this.location.translate(this.width/2, this.height/2);
   }
}
```

```
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);
   }
}
```

distanceFromCenterToO() test

```
public class RectangleTest extends TestCase {
   ...
   public void testdistanceFromCenterToO() {
      assertEquals(new Rectangle(new CartPt(3, 4), 4, 17)
        .distanceFromCenterToO(), 13.0, 0.001);
      Rectangle re = new Rectangle(new CartPt(1, 4), 7, 10);
      assertEquals(re.distanceFromCenterToO(), , 0.001));
   }
}
```

Class diagram



Rectangle

- CatesianPoint northWestConner
- int height
- int width

+ double distanceToO()
+ double distanceFromCenterToO()
+ CartPt center()

CartPt

- int x
- int y

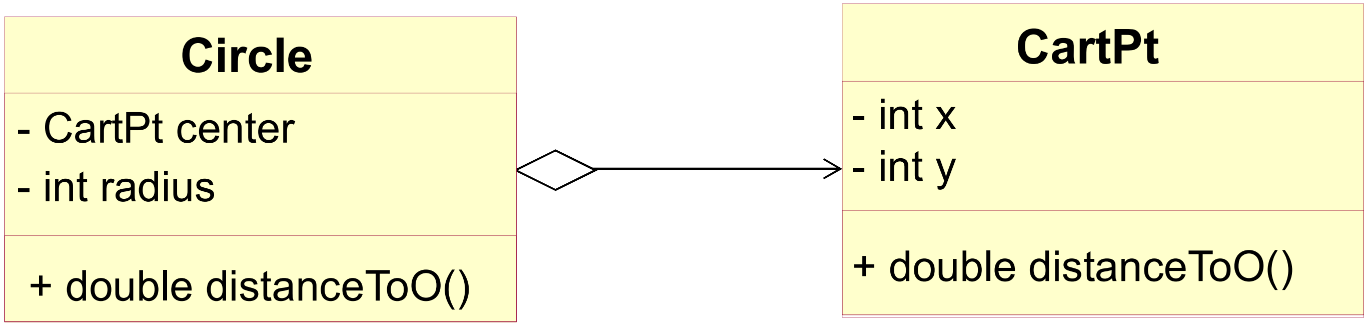
+ double distanceToO()
+ CartPt translate(int dx, int dy)

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 from circle to the origin



Circle

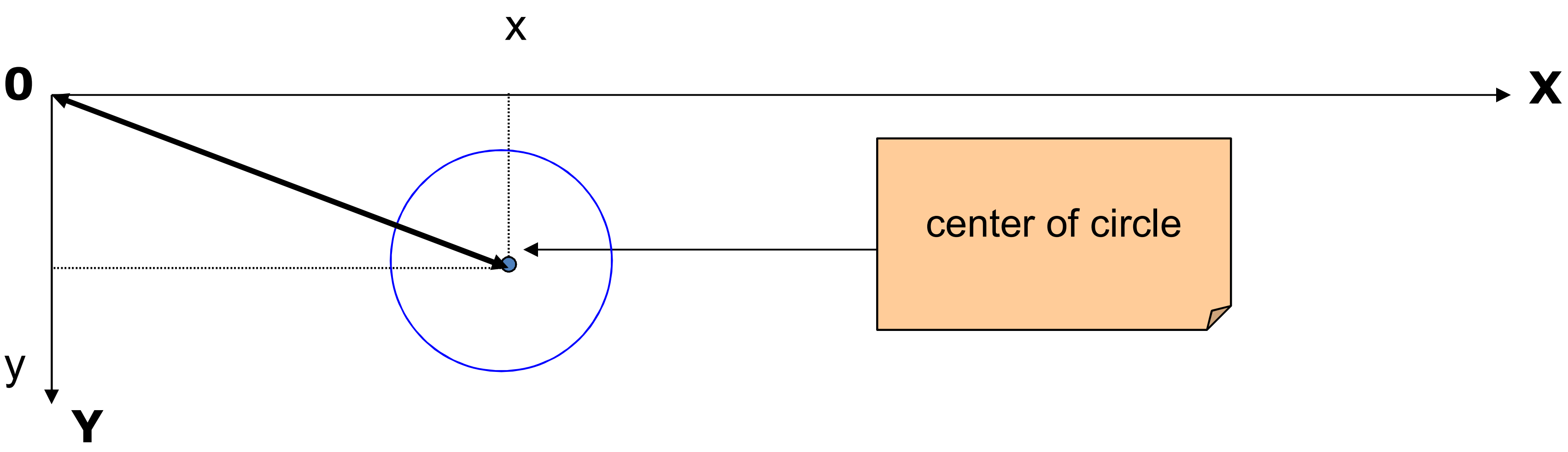
- CartPt center
- int radius

+ double distanceToO()

CartPt

- int x
- int y

+ double distanceToO()



center of circle

distanceToO template

```java
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() {
     ...this.center.distanceToO()...
     ...this.radius...
}
```

distanceToO body

```java
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 class CircleTest extends TestCase {
   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(-1, 2), 20);
      assertEquals(c1.distanceToO(), 5.0, 0.001);
      assertEquals(c2.distanceToO(), 13.0, 0.001);
      assertEquals(c3.distanceToO(), 2.236, 0.001);
   }
}
```

Computing the perimeter of a circle

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

```java
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 class CircleTest extends TestCase {
   ...
   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(-1, 2), 20);
      assertEquals(c1.perimeter(), 31.416, 0.001);
      assertEquals(c2.perimeter(), 62.832, 0.001);
      assertEquals(c3.perimeter(), 125.664, 0.001);
   }
}
```

Computing the area of a circle

area template

```java
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...
   }
}
```

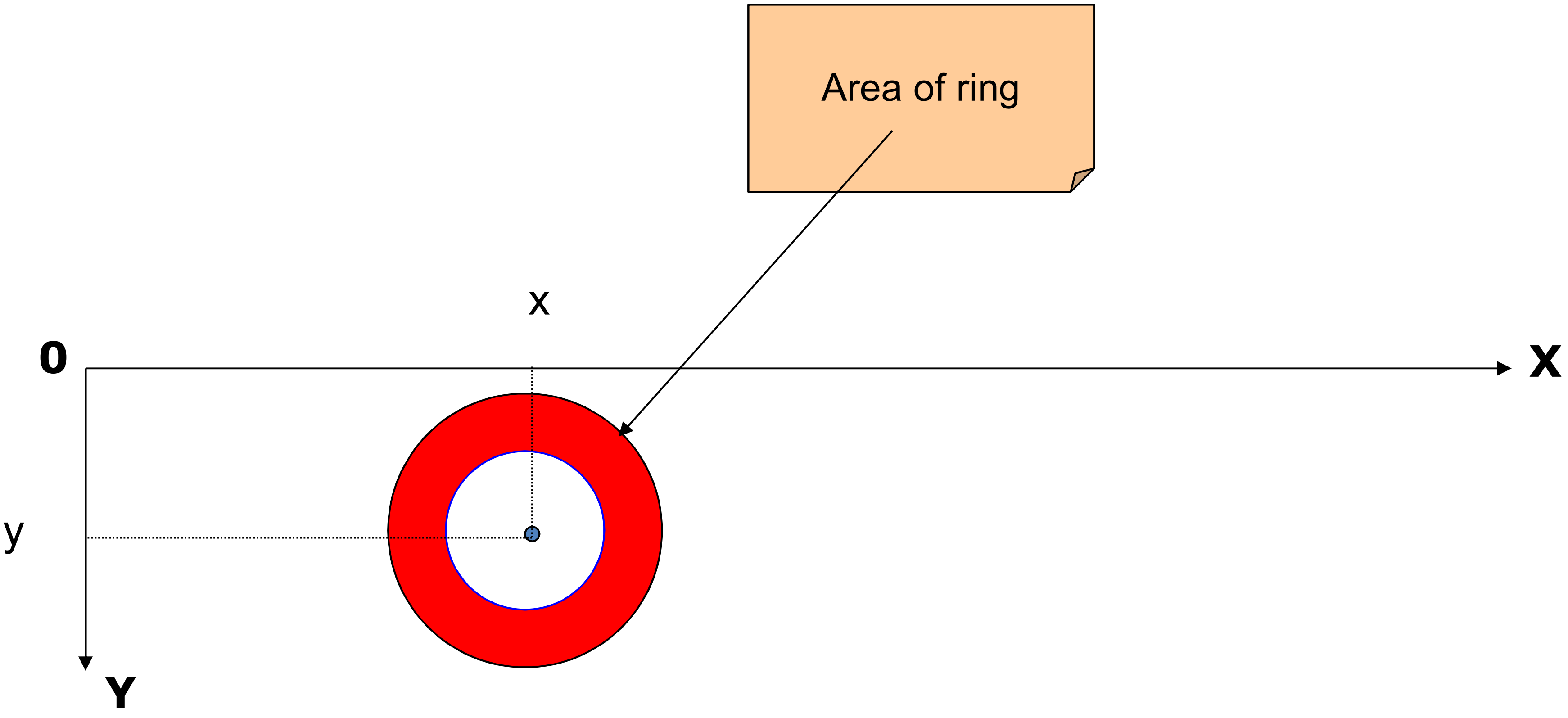
area body

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

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

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

area Test

```
public class CircleTest extends TestCase {
   ...
   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(-1, 2), 20);

      assertEquals(c1.area(), 78.539, 0.001);
      assertEquals(c2.area(), 314.159, 0.001);
      assertEquals(c3.area(), 1256.637, 0.001);
   }
}
```

Computes the area of a 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(int otherRadius) {
      ...this.center...this.radius...
      ...this.distanceToO()...this.perimeter()...this.area()...
      ... otherRadius...
  }
}
```

area of ring 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(int otherRadius) {
      double otherArea = Math.PI * otherRadius * otherRadius;
      return Math.abs(this.area() - otherArea);
   }
}
```

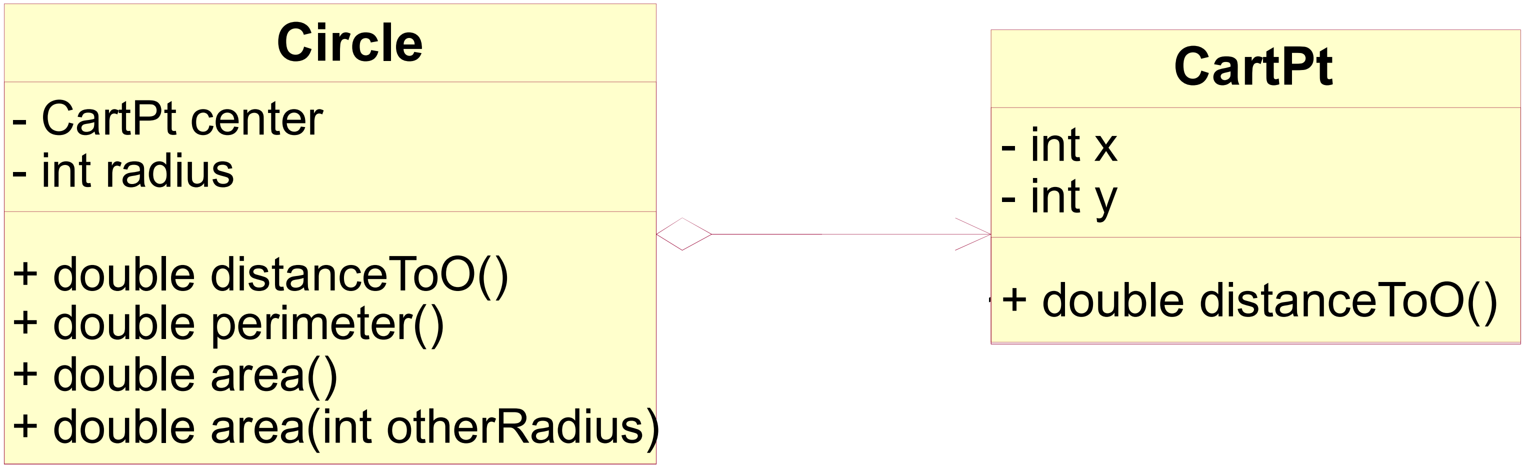
area of ring Test

```
public class CircleTest extends TestCase {
   ...
   public void testArea() {
      Circle c1 = new Circle(new CartPt(3, 4), 5);
      Circle c2 = new Circle(new CartPt(3, 4), 10);
      Circle c3 = new Circle(new CartPt(3, 4), 20);

      assertEquals(c1.area(), 78.539, 0.001);
      assertEquals(c2.area(), 314.159, 0.001);
      assertEquals(c3.area(), 1256.637, 0.001);

      assertEquals(c2.area(5), 235.619, 0.001);
      assertEquals(c3.area(5), 1178.097, 0.001);
      assertEquals(c3.area(10), 942.478, 0.001);
   }
}
```

Overloading method

- **Q**: what happen with the same name `area()` and `area(int)` method?
- **A:**
  - Method `area()` and `area(int)` 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



Circle

- CartPt center
- int radius

+ double distanceToO()
+ double perimeter()
+ double area()
+ double area(int otherRadius)

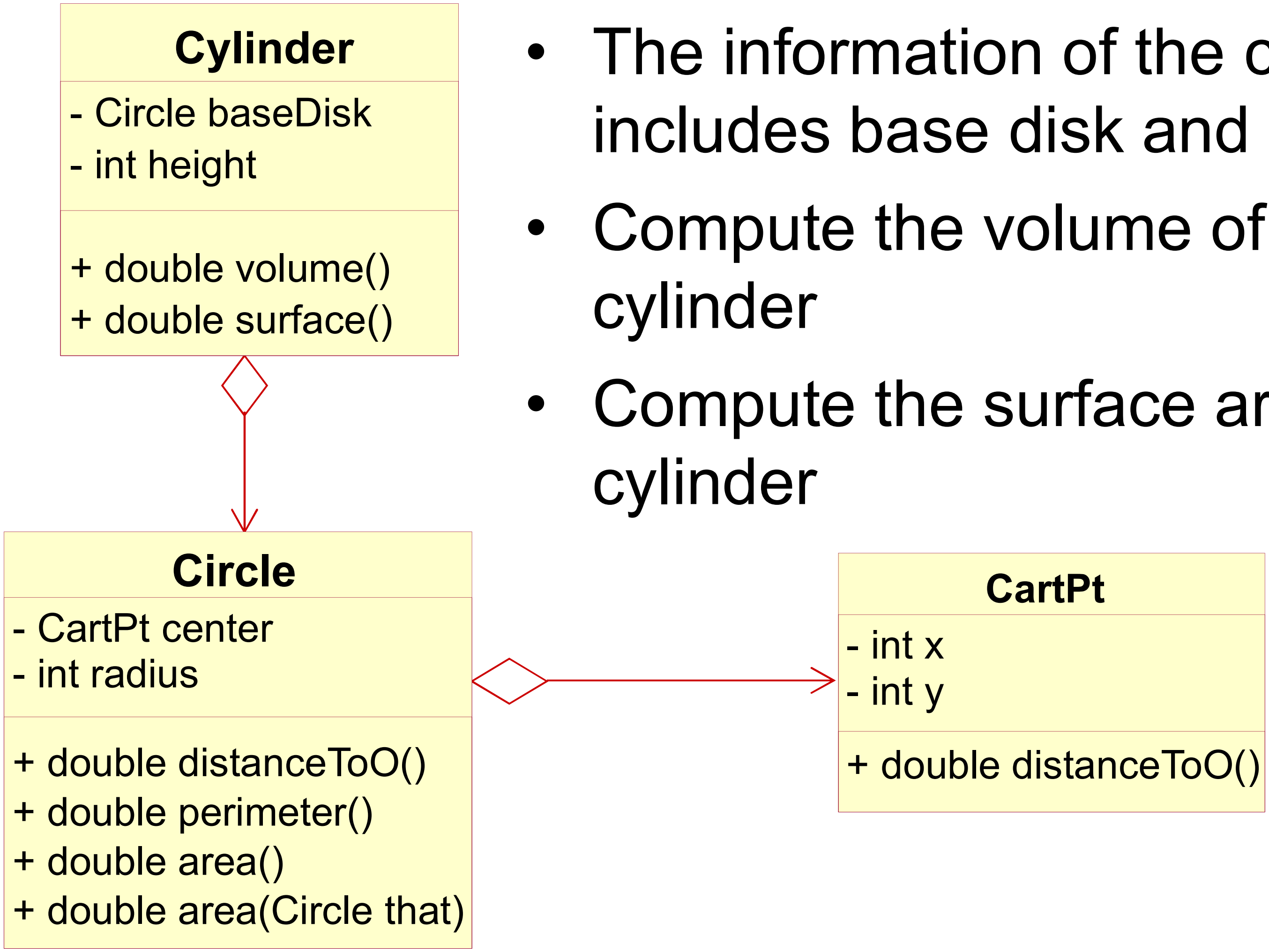
CartPt

- int x
- int y

+ double distanceToO()

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



Cylinder

- Circle baseDisk
- int height

+ double volume()
+ double surface()

Circle

- CartPt center
- int radius

+ double distanceToO()
+ double perimeter()
+ double area()
+ double area(Circle that)

CartPt

- int x
- int y

+ double distanceToO()

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

```java
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(), 785.398, 0.001);
   assertEquals(cy2.volume(), 9424.778, 0.001);
   assertEquals(cy3.volume(), 50265.482, 0.001);
}
```

`surface` method template

```java
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...
   }
}
```

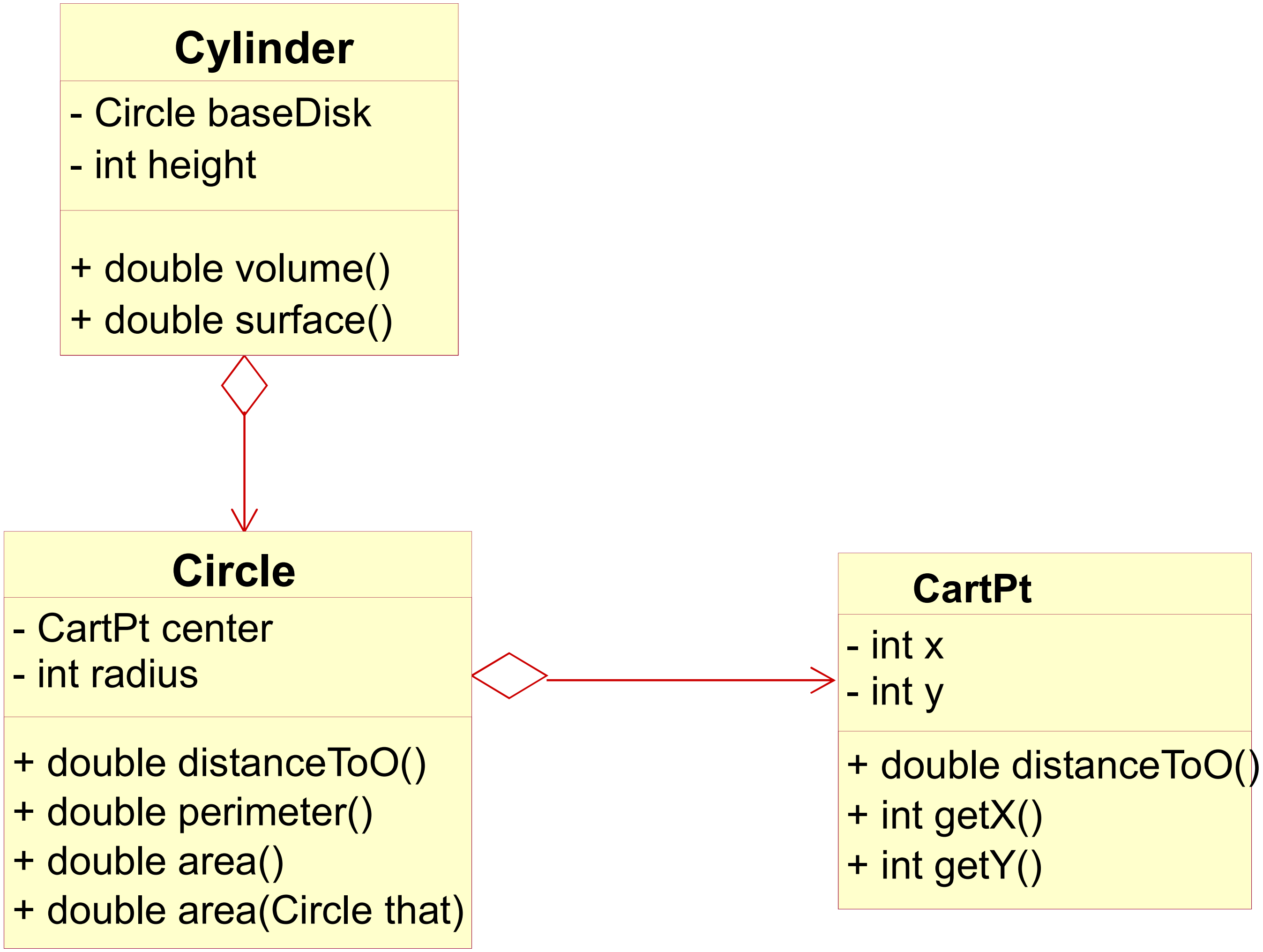
surface method body

```
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(), 314.159, 0.001);
      assertEquals(cy2.surface(), 1884.956, 0.001);
      assertEquals(cy3.surface(), 5026.548, 0.001);
   }
```

Class diagram



Cylinder

- Circle baseDisk
- int height

+ double volume()
+ double surface()

Circle

- CartPt center
- int radius

+ double distanceToO()
+ double perimeter()
+ double area()
+ double area(Circle that)

CartPt

- int x
- int y

+ double distanceToO()
+ int getX()
+ int getY()