Containment in Unions and Methods

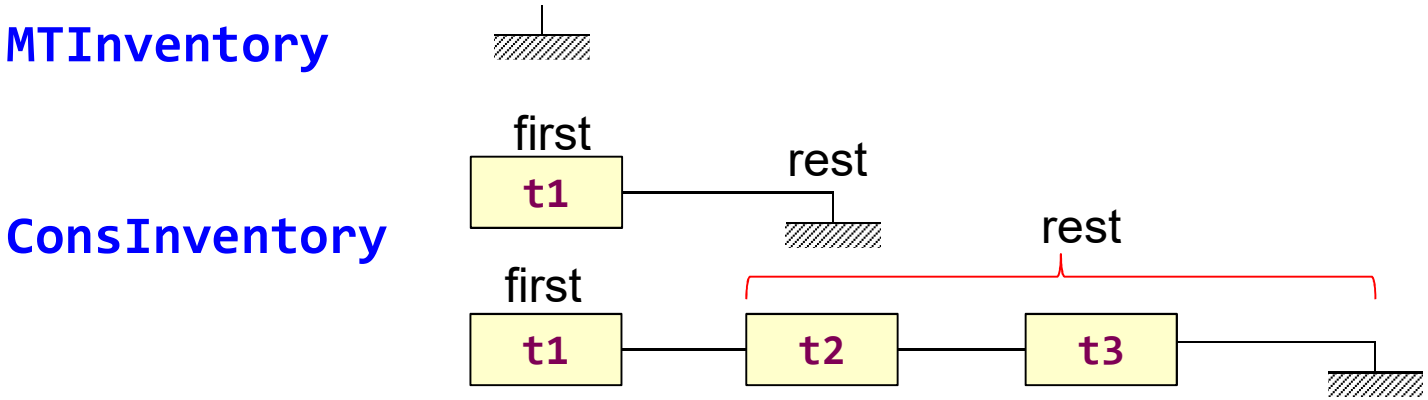
Part 1:
Containment in union
Self-References

Managing Inventory

- A sales clerk in a toy store needs to know not only the name of the toy, but also its price, warehouse availability.
- The representation of an inventory as a **list of toys**.

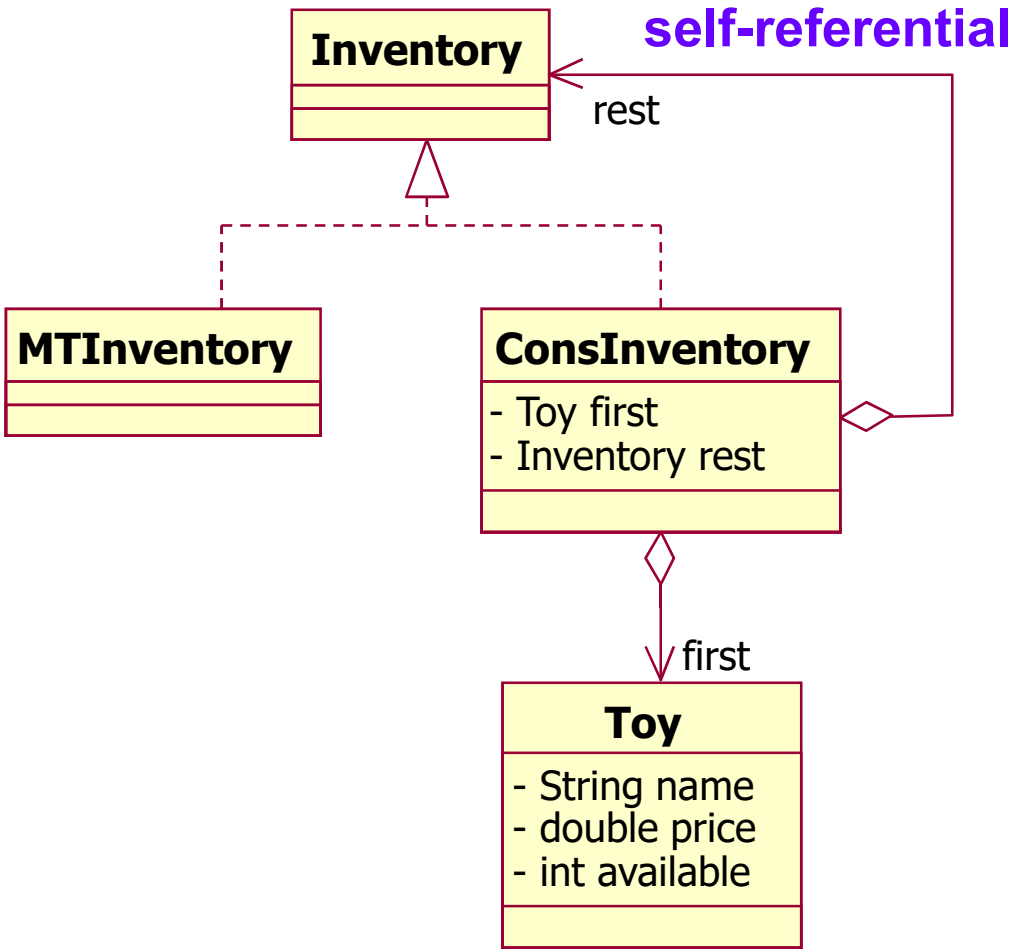
Data definition

- `Inventory` is one of:
  - a **empty**
  - a **construct** of `Toy Inventory`
- Inventory is a union:
  - `Inventory`, which is the type of **all kind of inventories;**
  - `MTInventory`, which represents an empty inventory; and
  - `ConsInventory`, which represents the **construction** of a new inventory from a `Toy` and an existing `Inventory`.



Class diagram

- An **`MTInventory`** class don't have any fields for it.
- A **`ConsInventory`** class requires two field definitions:
  - one for the first **`Toy`**
  - and one for the rest of the **`Inventory`**.



Define classes an constructors

```
public interface Inventory {
}
```

```
public class MTInventory implements Inventory {
}
```

```
public class ConsInventory implements Inventory {
   private Toy first;
   private Inventory rest;
   public ConsInventory(Toy first, Inventory rest) {
      this.first = first;
      this.rest= rest;
   }
}
```

Define classes and constructors

```
public class Toy {
   private String name;
   private double price;
   private int available;

   public Toy(String name, double price,
              int available) {
      this.name = name;
      this.price = price;
      this.available = available;
   }
}
```

Test Constructor

```
public class InventoryTest extends TestCase {
   public void testConstructor() {
      Toy doll = new Toy("doll", 17.95, 5);
      Toy robot = new Toy("robot", 22.05, 3);
      Toy gun = new Toy("gun", 15.0, 4);

      Inventory empty = new MTInventory();
      Inventory i1 = new ConsInventory(gun, empty);
      Inventory i2 = new ConsInventory(robot, i1);
      Inventory i3 = new ConsInventory(doll, i2);
      System.out.println(i3);

      Inventory all = new ConsInventory(doll,
             new ConsInventory(robot,
             new ConsInventory(gun, new MTInventory())));
      System.out.println(all);
   }
}
```

Print the content of an inventory

Q: How can we print the content of an object.

A: Use

```
System.out.println(object)
```

```
= System.out.println(object.toString())
```

overiding `toString()` method of class `Object`.

Q: Do we need to add `toString()` in `Inventory` interface?

A: No !

toString() in classes

```java
// inside of MTInventory class
public String toString() {
   return "";
}
```

```java
// inside of ConsInventory class
public String toString() {
   return this.first.toString() + "\n"
        + this.rest.toString();
}
```

```java
// inside of Toy class
public String toString() {
   return "name: " + this.name
        + ", price: " + this.price
        + ", available: " + this.available;
  }
}
```

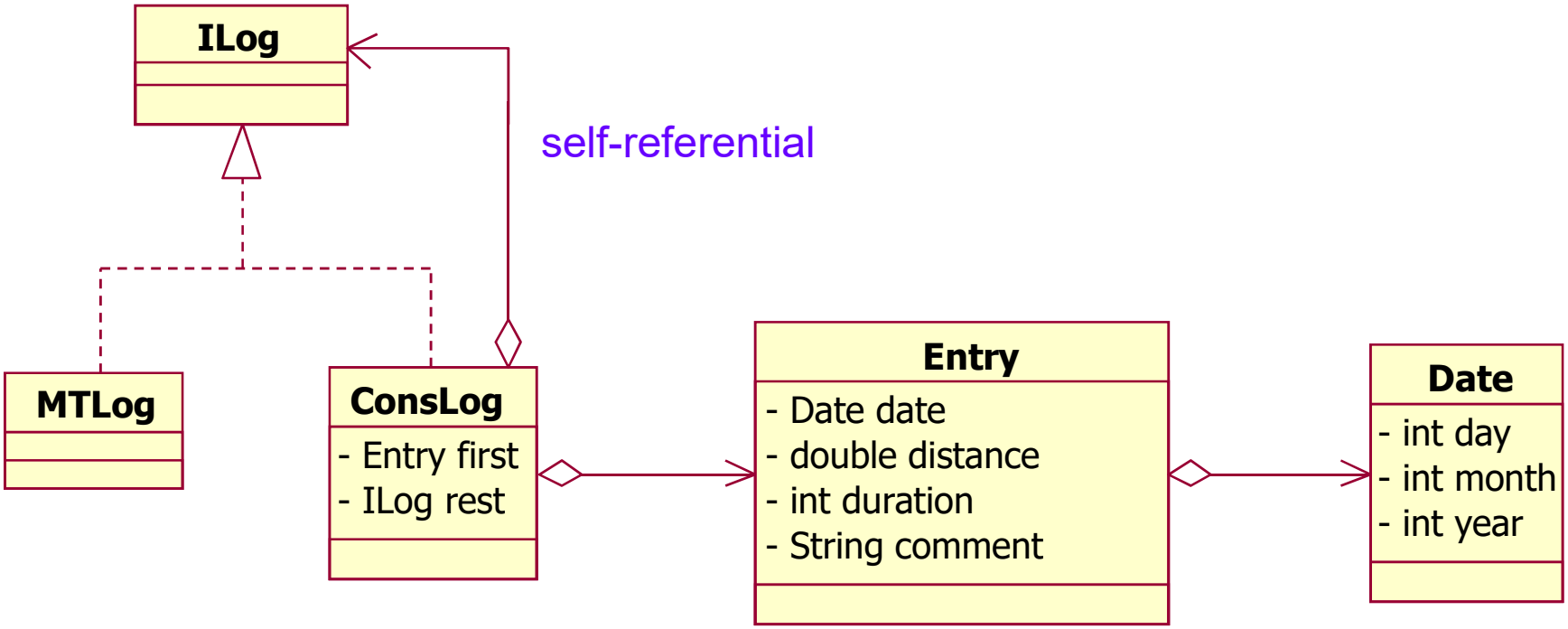
Managing a Runner’s Logs Example

- 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 disposition.
- Naturally the program shouldn't just deal with a single log entry but sequences of log entries.

Data definition

- The class of Logs is a union:
  - `ILog`, which is the type of all logs;
  - `MTLog`, which represents an empty log; and
  - `ConsLog`, which represents the construction of a new log from an entry and an existing log.

Class diagram



Define classes an constructors

```
public interface ILog {
}
```

```
public class MTLog implements ILog {
}
```

```
public class ConsLog implements ILog {
   private Entry first;
   private ILog rest;
   public ConsLog(Entry first, ILog rest) {
      this.first = first;
      this.rest = rest;
   }
}
```

Define classes and constructors

```
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;
   }
}
```

Define classes and constructors

```
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

```
public class LogTest extends TestCase {
   public void testConstructor() {
      Entry e1 =
          new Entry(new Date(5, 5, 2005), 5.0, 25, "Good");
      Entry e2 =
          new Entry(new Date(6, 6, 2005), 3.0, 24, "Tired");
      Entry e3 =
          new Entry(new Date(23, 6, 2005), 26.0, 156, "Great");

      ILog empty = new MTLog();
      ILog l1 = new ConsLog(e3, empty);
      ILog l2 = new ConsLog(e2, l1);
      ILog l3 = new ConsLog(e1, l2);
      System.out.println(l3);

      ILog all = new ConsLog(e1, new ConsLog(e2,
                     new ConsLog(e3, new MTLog())));
      assertEquals(l3, all);
   }
}
```

toString() method

```
// inside of MTLog class
public String toString() {
   return "";
}
```

```
// inside of ConsLog class
public String toString() {
   return this.first.toString() + " \n" + this.rest.toString();
}
```

```
// inside of Entry class
public String toString() {
   return "date: " + this.date.toString()
        + ", distance: " + this.distance
        + ", duration: " + this.duration
        + ", comment: " + this.comment;
}
```

```
// inside of Date class
public String toString() {
   return this.day + "/" + this.month + "/" + this.year;
}
```

equals() method

```
// in MTLog class
public boolean equals(Object obj) {
   if (obj == null || !(obj instanceof MTLog)
      return false;
   return true;
}
```

```
// inside ConsLog class
public boolean equals(Object obj) {
      if (obj == null || !(obj instanceof ConsLog))
         return false;
      else {
         ConsLog that = (ConsLog) obj;
         return this.first.equals(that.first)
               && this.rest.equals(that.rest);
      }
   }
```

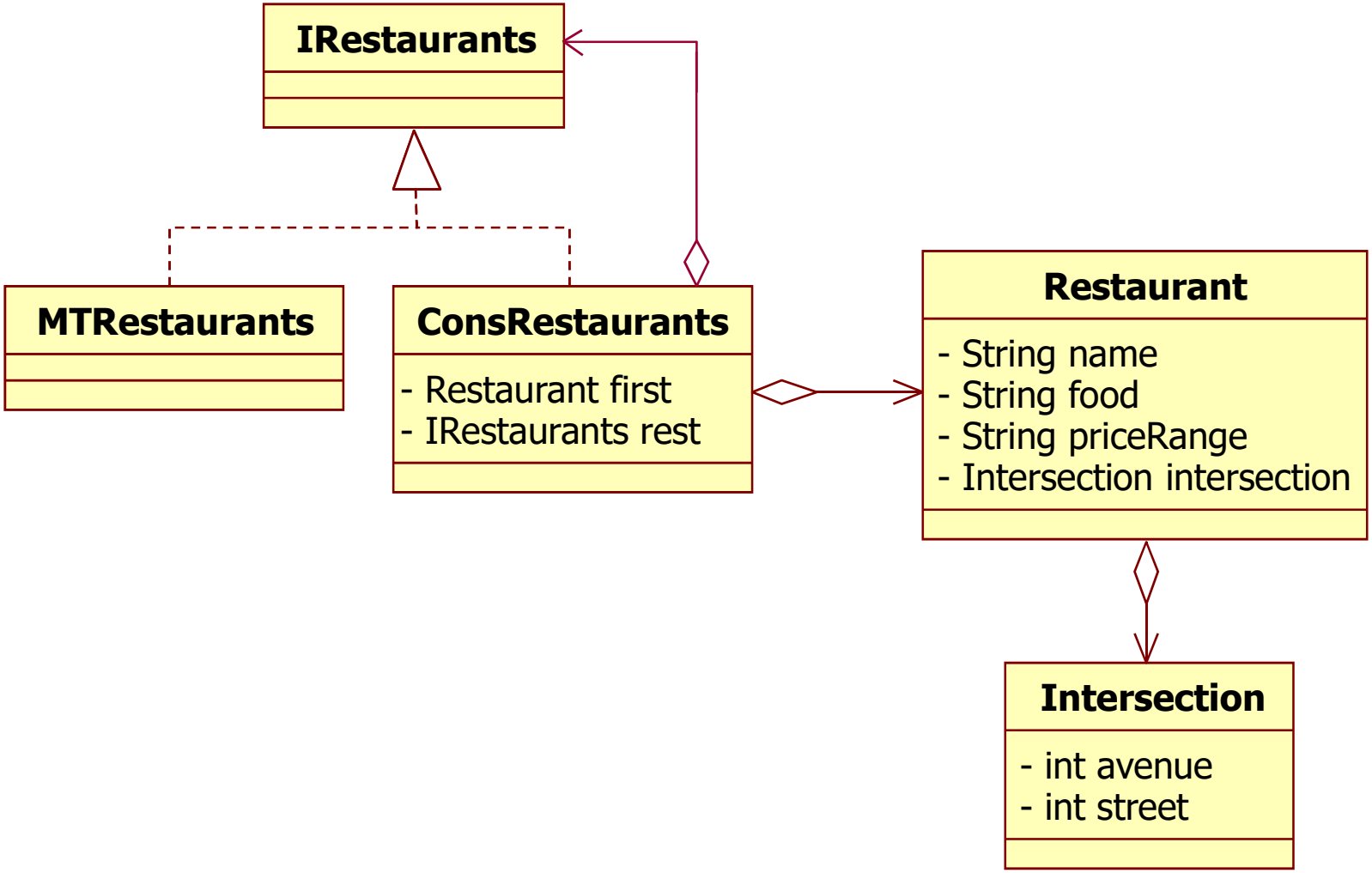
```java
// in Entry class
public boolean equals(Object obj) {
   if (obj == null || !(obj instanceof Entry))
      return false;
   else {
      Entry that = (Entry) obj;
      return this.date.equals(that.date) &&
          this.distance == that.distance &&
          this.durationInMinutes == that.durationInMinutes &&
          this.postRunFeeling.equals(that.postRunFeeling);
      }
   }
}
```

```java
// inside Date class
public boolean equals(Object obj) {
   if (obj == null || !(obj instanceof Date))
      return false;
   else {
      Date that = (Date) obj;
      return this.day == that.day &&
             this.month == that.month &&
             this.year == that.year;
   }
}
```

Recall 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).
- Clearly, the visitor assistant should deal with lists of restaurants, not just individual restaurants. A visitor may, for example, wish to learn about all Chinese restaurants in a certain area or all German restaurants in a certain price range.

Class diagram



Define classes and constructors

```
public interface IRestaurants {
}
```

```
public class MTRestaurants implements IRestaurants {
}
```

```
public class ConsRestaurants implements IRestaurants {
  private Restaurant first;
  private IRestaurants rest;
  public ConsRestaurants(Restaurant first,
                         IRestaurants rest) {
    this.first = first;
    this.rest = rest;
  }
}
```

Define Restaurant class

```
public class Restaurant {
   private String name;
   private String food;
   private String priceRange;
   private Intersection intersection;
   public Restaurant(String name, String food,
                     String priceRange,
                     Intersection intersection) {
      this.name = name;
      this.food = food;
      this.priceRange = priceRange;
      this.intersection = intersection;
   }
}
```

Define `Intersection` class

```
public class Intersection {
   private int avenue;
   private int street;

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

toString() method

```
// in class ConsRestaurants
public String toString() {
  return this.first.toString() + " \n" + this.rest.toString();
}
```

```
// in class MTRestaurants
public String toString() {
  return "";
}
```

```
// in class Restaurant
public String toString() {
  return "Name: " + this.name + ", food: " + this.food
     + ", range price: " + this.priceRange
     + ", intersection: " + this.intersection.toString() + "\n"
     + this.rest;
  }
}
```

```
// in class Intersection
public String toString() {
  return "avenue: " + this.avenue
       + ", street: " + this.street;
}
```

equals() method

```
// in MTRestaurants class
public boolean equals(Object obj) {
   if (obj == null || !(obj instanceof MTRestaurants)
      return false;
   return true;
}
```

```
// inside ConsRestaurants class
public boolean equals(Object obj) {
      if (obj == null || !(obj instanceof ConsRestaurants))
         return false;
      else {
         ConsRestaurants that = (ConsRestaurants) obj;
         return this.first.equals(that.first)
               && this.rest.equals(that.rest);
      }
   }
```

```java
// in Restaurants class
public boolean equals(Object obj) {
   if (obj == null || !(obj instanceof Restaurant))
      return false;
   else {
      Restaurant that = (Restaurant) obj;
      return this.name.equals(that.name) &&
          this.food.equals(that.food) &&
          this.priceRange.equals(that.priceRange) &&
          this.intersection.equals(that.intersection);
      }
   }
}
```

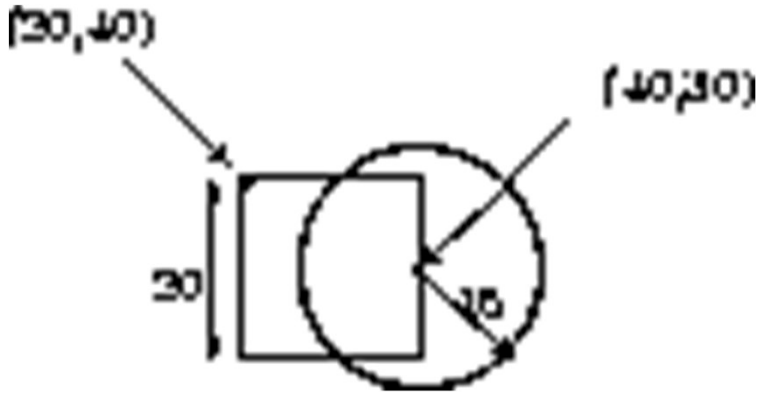
```java
// inside Intersection class
public boolean equals(Object obj) {
   if (obj == null || !(obj instanceof Intersection))
      return false;
   else {
      Intersection that = (Intersection) obj;
      return this.avenue == that.avenue &&
             this.street == that.street;
   }
}
```

Test Constructor

```
public class RestaurantsTest extends TestCase {
  public void testConstructor() {
    Restaurant r1 = new Restaurant("Chez Nous",
         "French", "exp.", new Intersection(7, 65));
    Restaurant r2 = new Restaurant("Das Bier",
         "German", "cheap", new Intersection(2, 86));
    Restaurant r3 = new Restaurant("Sun",
       "Chinese", "cheap", new Intersection(10, 13));

    IRestaurants empty = new MTRestaurants();
    IRestaurants l1 = new ConsRestaurants(r3, empty);
    IRestaurants l2 = new ConsRestaurants(r2, l1);
    IRestaurants l3 = new ConsRestaurants(r1, l2);
    System.out.println(l3);

    IRestaurants all = new ConsRestaurants(r1,
        new ConsRestaurants (r2,
        new ConsRestaurants(r3, new MTRestaurants())));
    assertEquals(all, l3);
  }
}
```

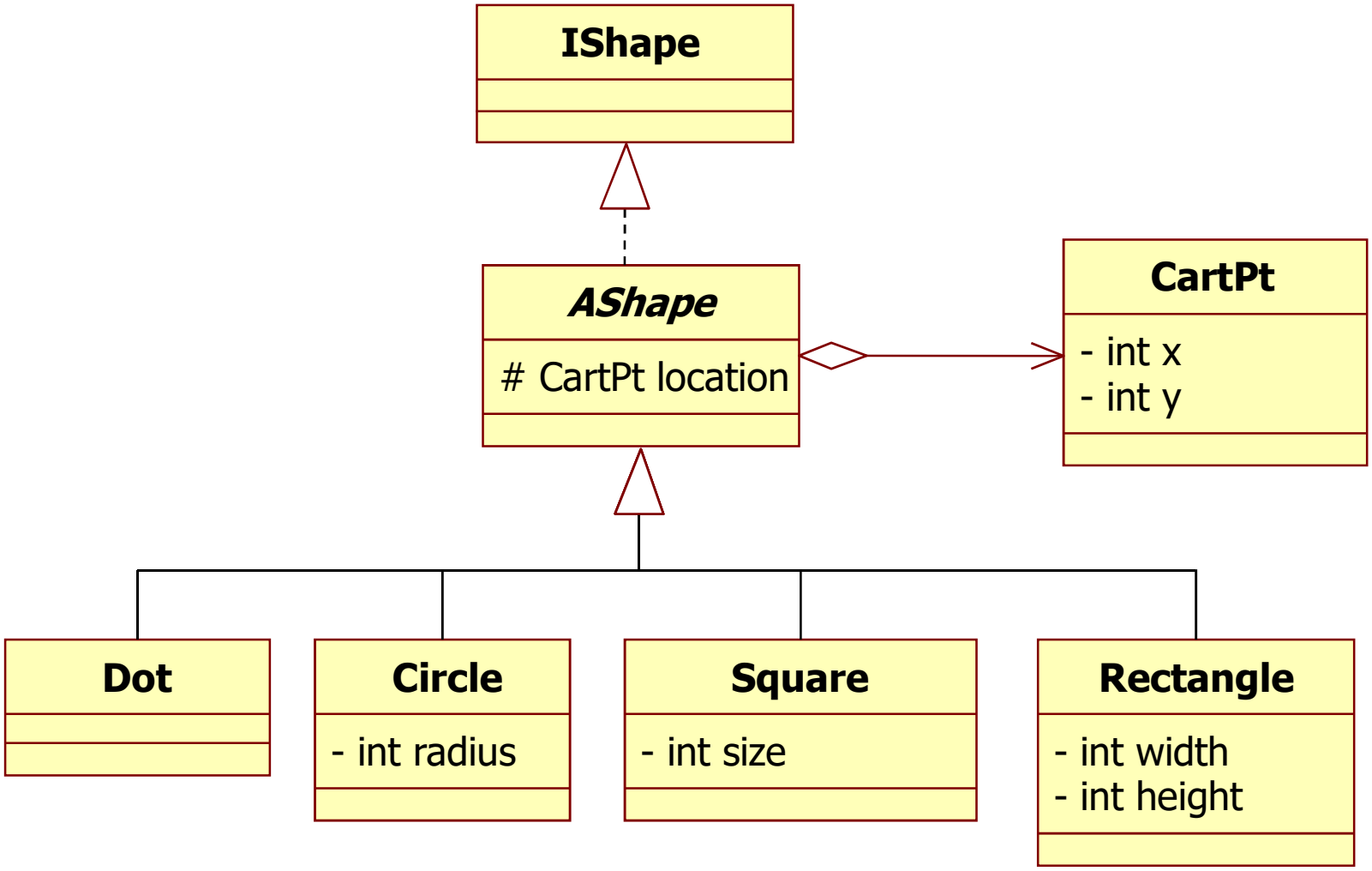
Overlaying shape example

- Develop a drawing program that deals with at least three kinds of shapes: dots, squares, and circles. In addition, the program should also deal with ***overlaying shapes*** on each other. In the following figure, for example, we have superimposed a circle on the right side of a square:

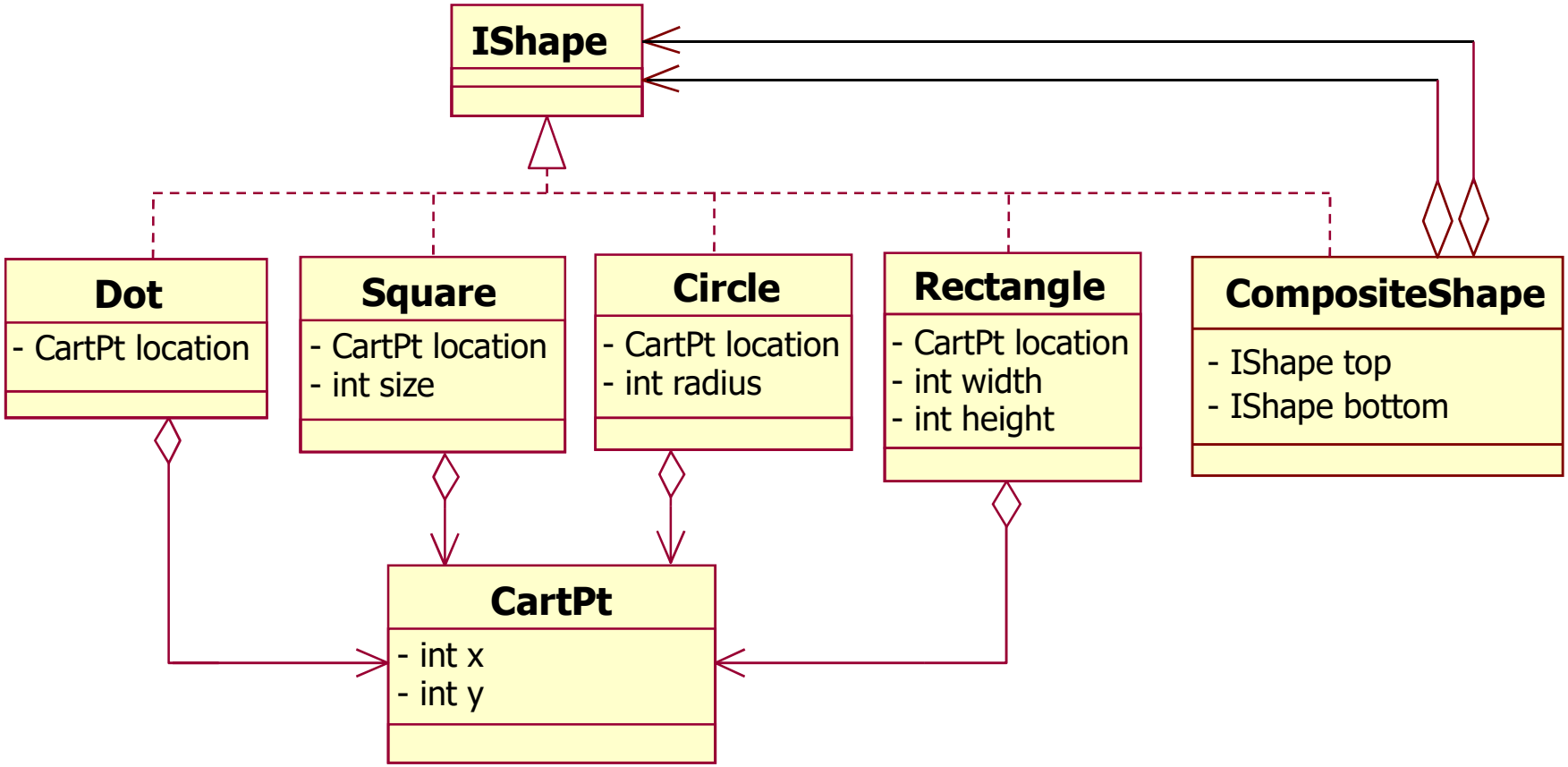


- We could now also superimpose this compounded shape on another shape and so on.

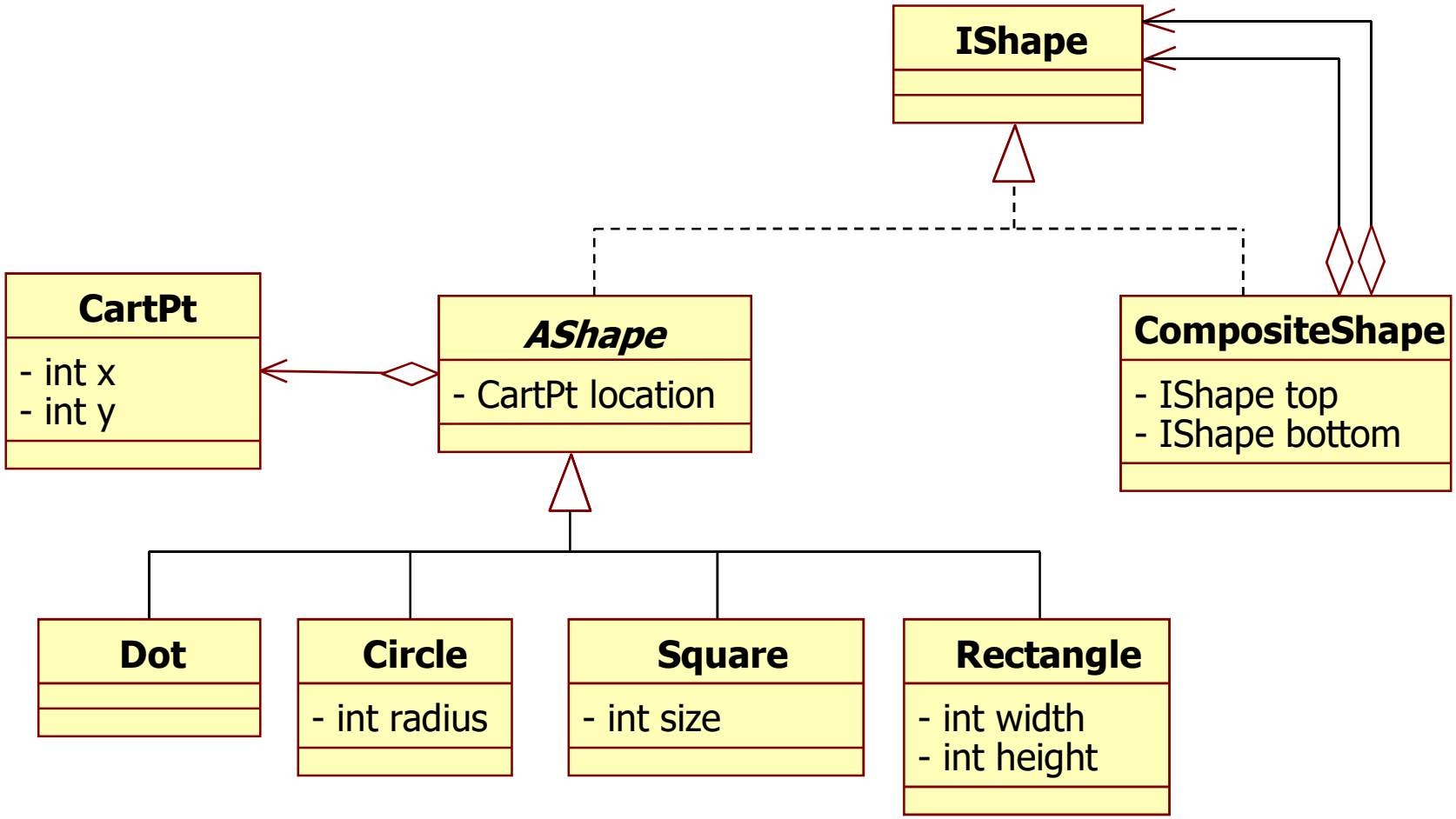
Old class design



New design after add Composite Shape



New design after add Composite Shape



Define classes and constructors

```
public interface IShape {
}
```

```
public class CompositeShape implements IShape {
   private IShape top;
   private IShape bottom;
   public CompositeShape(IShape top, IShape bottom) {
      this.top = top;
      this.bottom = bottom;
   }
}
```

```
public abstract class AShape implements IShape {
   protected CartPt location;
   public ASingleShape(CartPt location) {
      this.location = location;
   }
}
```

Define classes and constructors

```
public class Square extends AShape {
  private int size;
  public Square(CartPt location, int size){
    super(location);
    this.size = size;
  }
}
```

```
public class Circle extends AShape {
  private int radius;
  public Circle(CartPt location, int radius) {
    super(location);
    this.radius = radius;
  }
}
```

```
public class Dot extends AShape {
  public Dot(CartPt location) {
    super(location);
  }
}
```

Define classes and constructors

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

Test Constructor

```
public class ShapeTest extends TestCase {
   public void testConstructor() {
      IShape s1 = new Square(new CartPt(4, 3), 40);
      IShape s2 = new Square(new CartPt(3, 4), 50);
      IShape c1 = new Circle(new CartPt(0, 0), 20);
      IShape c2 = new Circle(new CartPt(12, 5), 20);

      IShape u1 = new CompositeShape(s1, s2);
      IShape u2 = new CompositeShape(s1, c2);
      IShape u3 = new CompositeShape(c1, u1);
      IShape u4 = new CompositeShape(u3, u2);
      IShape u5 = new CompositeShape(s1,
                     new Compositeshape(c1, s2));
      System.out.println(u5);
   }
}
```

toString() method

```
// in class CompositeShape
public String toString() {
  return this.top.toString() + " \n" + this.bottom.toString();
}
```

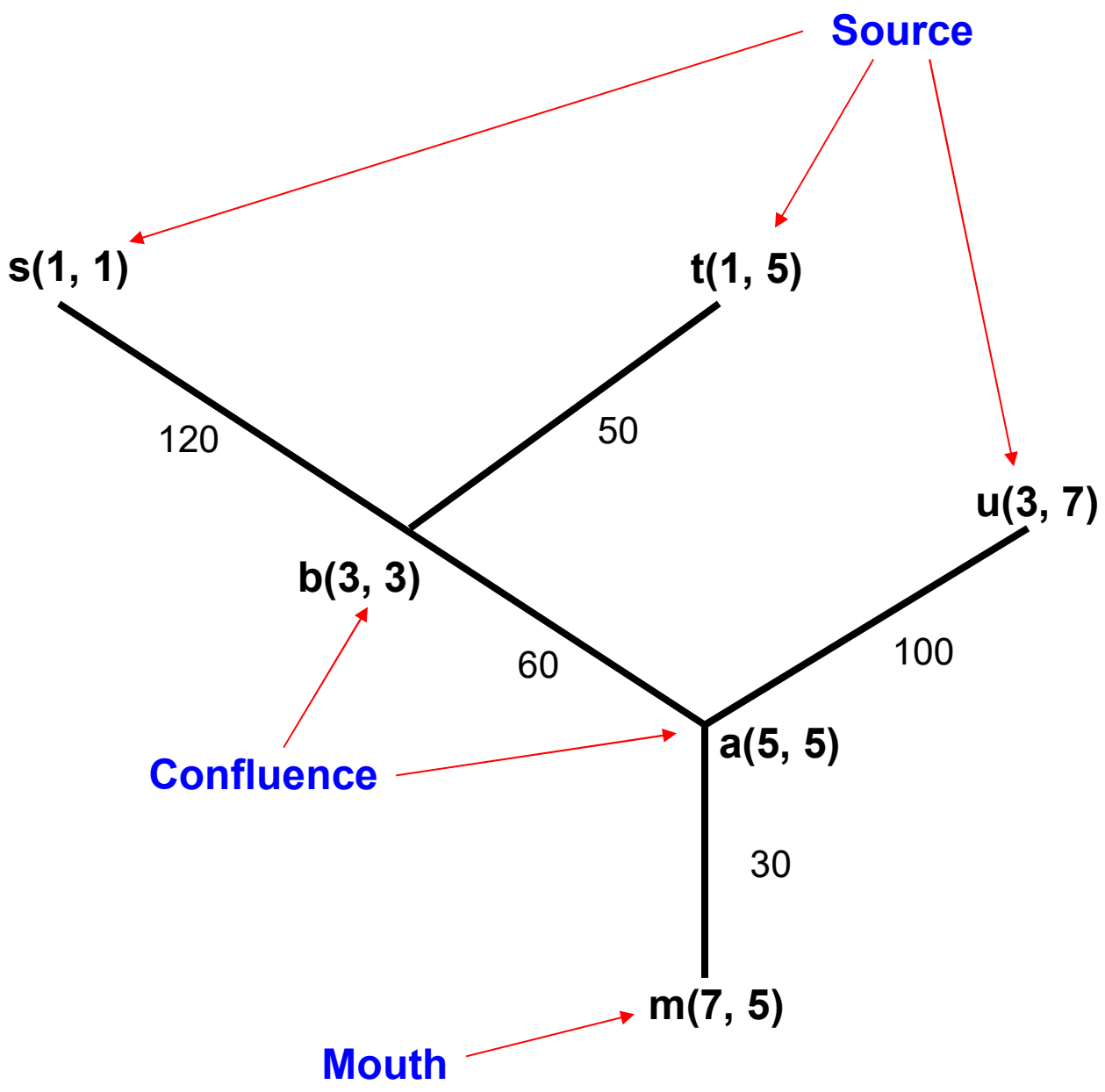
```
// in class ASingleShape
public String toString() {
  return "Location: " + this.location.toString();
}
```

```
// in class Square
public String toString() {
  return "Square(" + super.toString() + "Size: " + this.size + ")";
}
```

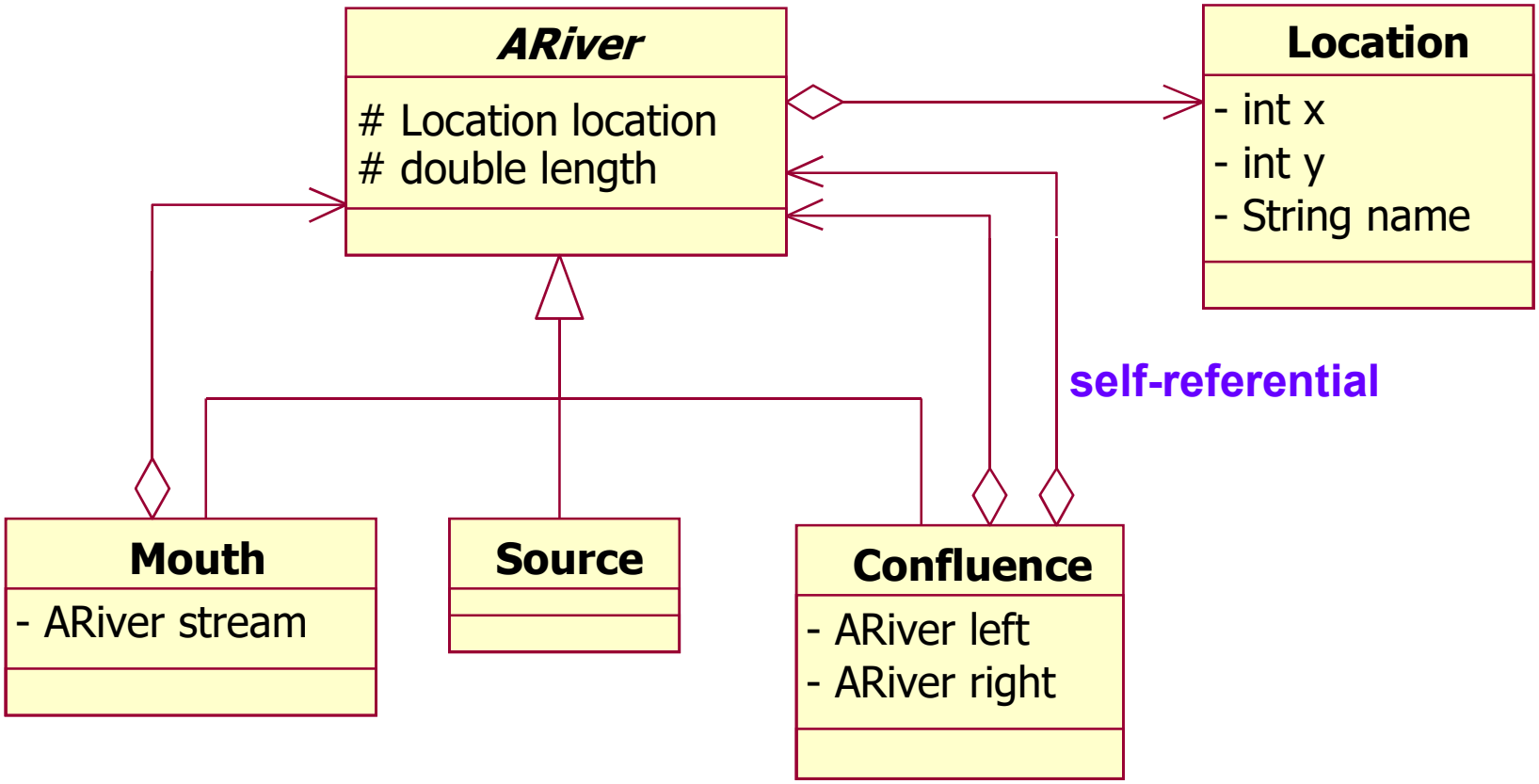
```
// in class CartPt
public String toString() {
  return "CartPt(" + this.x + ", " + this.y + ")";
}
```

River Systems Example

- The environmental protection agency monitors the water quality for river systems.
- A river system consists of a source of river, its tributaries (nhánh sông), the tributaries of the tributaries, and so on. Besides, each of part in the river system has location, and its length.
- The place where a tributary flows into a river is called confluence (hợp dòng).
- The initial river segment is its source (nguồn)
- The river's end - the segment that ends in a sea or another river - is called its mouth (cửa sông)

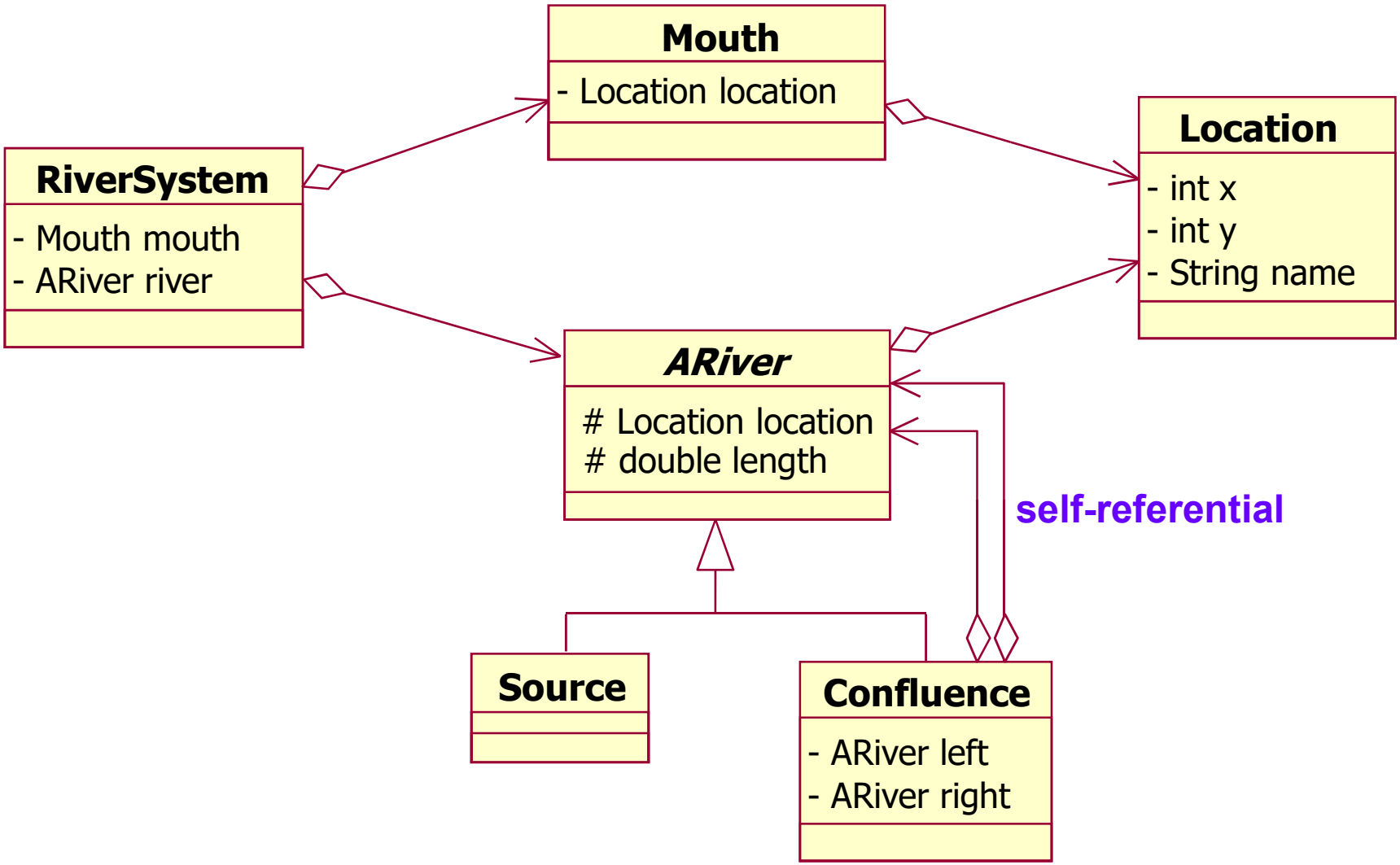

Source
s(1, 1)
t(1, 5)
120
50
u(3, 7)
b(3, 3)
60
100
Confluence
a(5, 5)
30
m(7, 5)
Mouth

Class Diagram (design 1)

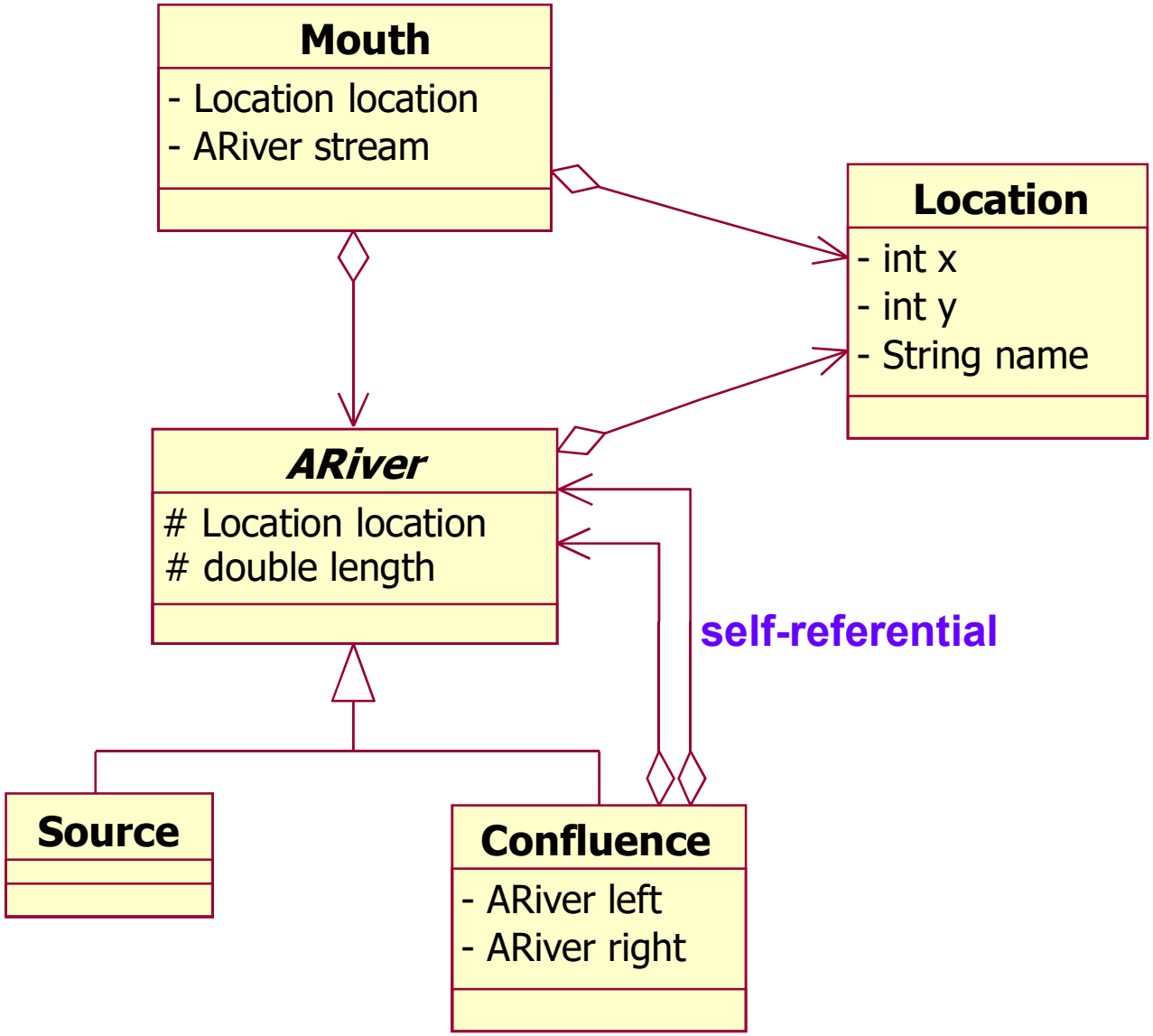


Mouth have not the length info

Class diagram (design 2)



Class diagram (design 2)



Define classes and constructors

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

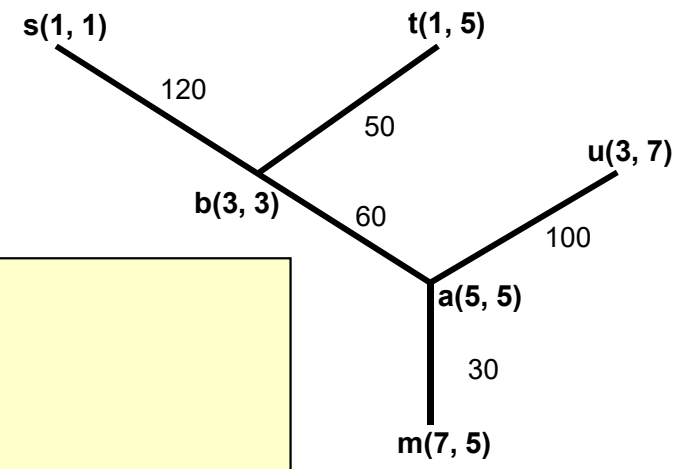
```
public class Mouth {
   private Location location;
   private ARiver stream;
   public Mouth(Location location, ARiver stream) {
      this.location = location;
      this.stream = stream;
   }
}
```

```java
public abstract class ARiver {
   protected Location location;
   protected double length;
   public ARiver(Location location, double length) {
      this.location = location;
      this.length = length;
   }
}
```

```java
public class Source extends ARiver {
   public Source(Location location, double length) {
      super(location,length);
   }
}
```

```java
public class Confluence extends ARiver {
   private ARiver left;
   private ARiver right;
   public Confluence(Location location, double length,
                     ARiver left, ARiver right) {
      super(location,length);
      this.left = left;
      this.right = right;
   }
}
```

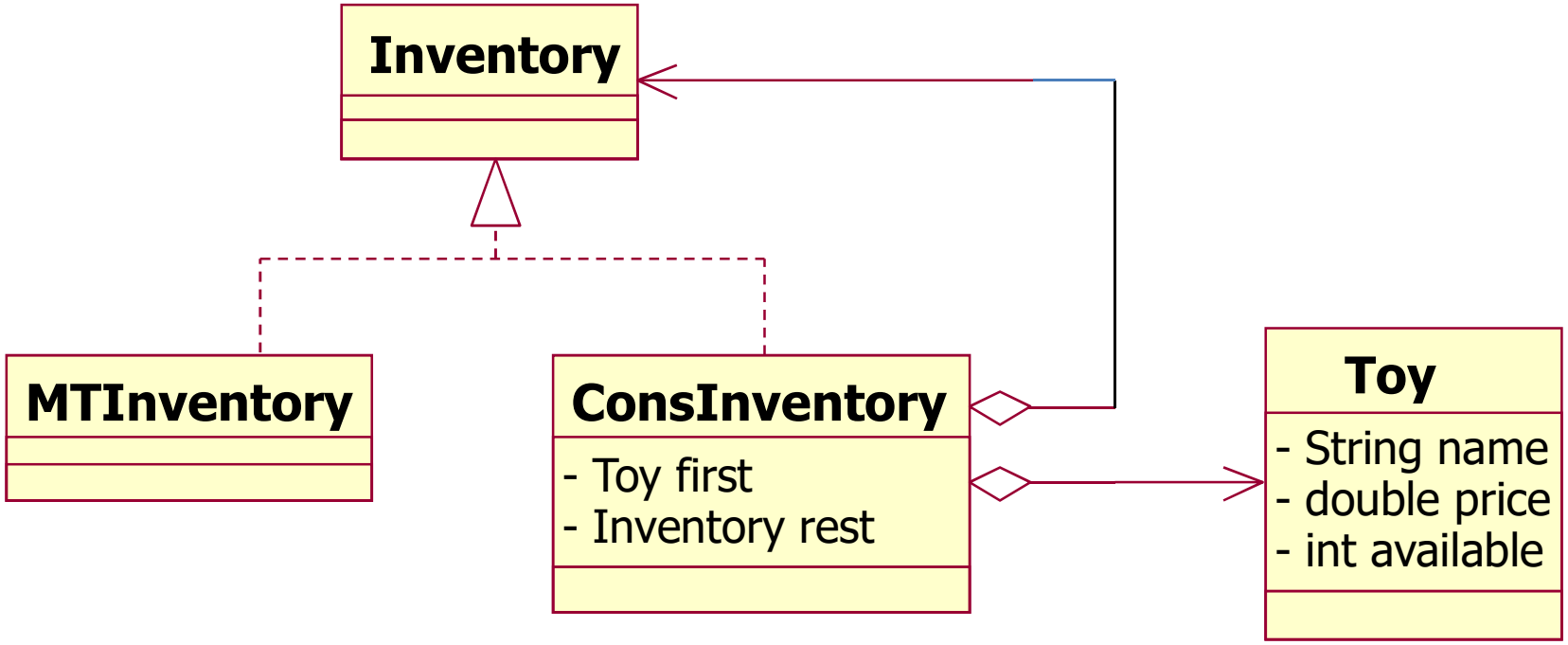
Test Constructor




```
public class ARiverTest extends TestCase {
   public void testConstructor() {
      ARiver s = new Source(
           new Location(1, 1, "s"), 120.0);
      ARiver t = new Source(
                 new Location(1, 5, "t"), 50.0);
      ARiver u = new Source(
           new Location(3, 7, "u"), 100.0);

      ARiver b = new Confluence(
           new Location(3, 3, "b"), 60.0, s, t);
      ARiver a = new Confluence(
           new Location(5, 5, "a"), 30.0, b, u);

      Mouth m = new Mouth(new Location(7, 5, "m"), a);
   }
}
```

Part 2: Methods and Classes with Self References

Recall Inventory problem

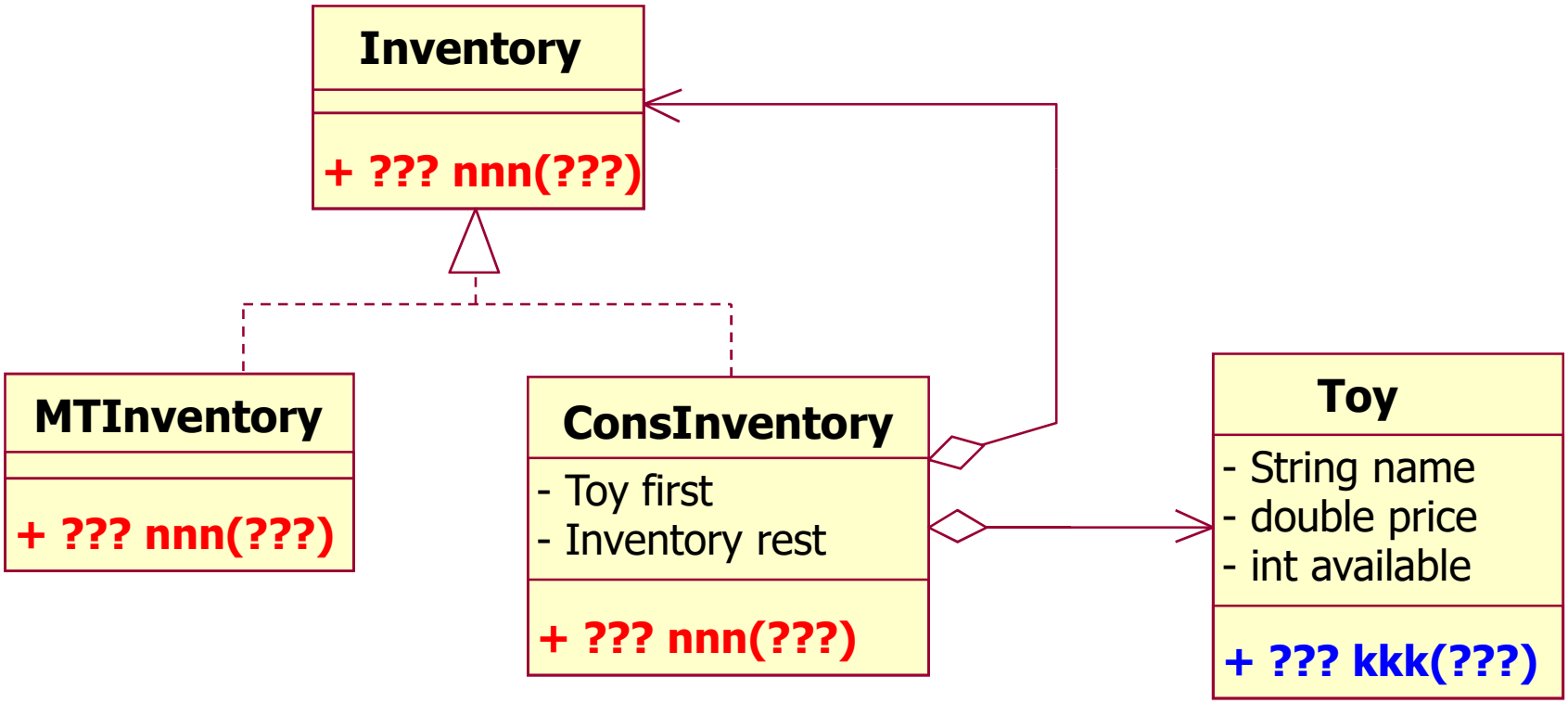


Class diagram

Recall Inventory problem

- Develop the method `contains`, which determines whether or not the name of toy occurs in the inventory
- Develop the method `isBelow`, which checks whether all of the prices of toys in inventory are below the threshold.
- Develop the method `howMany`, which produces the number of toys in the inventory.
- Develop the method `raisePrice`, which produces an inventory in which all prices are raised by a rate 5% (use ***mutable*** and ***immutable***).

Add methods to the `Inventory`'s Class Diagram



Q: Write Java method templates for all the classes in the class diagram ?

Java template for Toy

```
public class Toy {
   private String name;
   private double price;
   private int available;
   public Toy(String name, double price, int available) {
      this.name = name;
      this.price = price;
      this.available = available;
   }

   public ??? kkk(???) {
      ...this.name...
      ...this.price...
      ...this.available...
   }
}
```

Java template for `Inventory`

```
public interface Inventory {
   public ??? nnn(???);
}
```

Java template for `MTInventory`

```
public class MTInventory implements Inventory {

   public MTInventory () { }

   public ??? nnn(???) {
      ...
   }
}
```

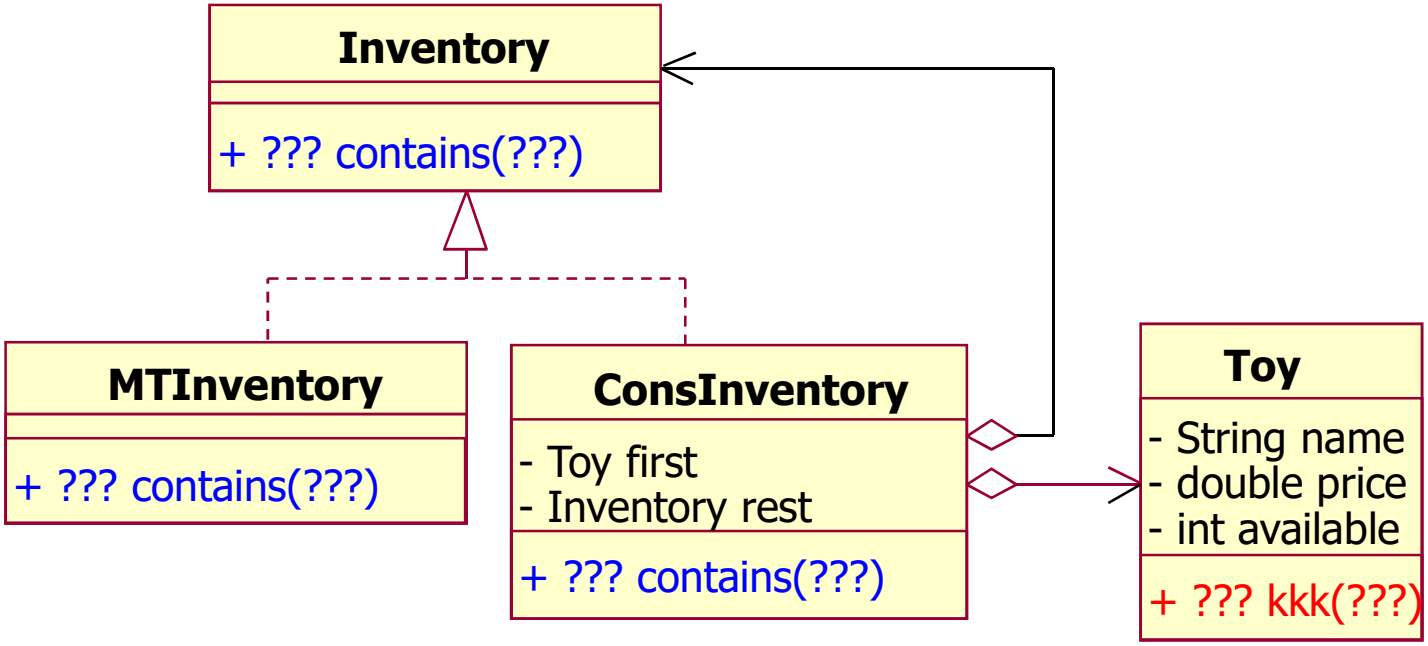
Java template for `ConsInventory`

```
public class ConsInventoy implements Inventory {
   private Toy first;
   private Inventory rest;
   public Cons(Toy first, Inventory rest) {
      this.first = first;
      this.rest = rest;
   }

   public ??? nnn(???) {
      ...this.first.kkk(???)...
      ...this.rest.nnn(???)...
   }
}
```

Since all instances in the **`rest`** field are always created from either **`MTInventory`** or **`ConsInventory`**, this means that the method call **`this.rest.nnn()`** really invokes one of the concrete **`nnn()`** methods in **`MTInventory`** or **`ConsInventory`**

Add `contains` method

- Develop the method contains, which determines whether or not the name of toy occurs in the Inventory



Purpose and contract of `contains()` for `Inventory`

```
public interface Inventory {
   // determines whether or not the name of
   // toy occurs in the Inventory
   public boolean contains(String toyName);
}
```

Examples to test `contains()`

```
Toy doll = new Toy("doll", 17.95, 5);
Toy robot = new Toy("robot", 22.05, 3);
Toy gun = new Toy ("gun", 15.0, 4);

Inventory empty = new MTInventory();
Inventory i1 = new ConsInventory(doll, empty);
nventory i2 = new ConsInventory(robot, i1);
Inventory all = new ConsInventory(doll,
                new ConsInventory(robot,
                new ConsInventory(gun, new MTInventory())));

empty.contains("robot") → should be false
i1.contains("robot") → should be false
i2.contains("robot") → should be true
all.contains("robot") → should be true
all.contains("car") → should be false
```

contains() for MTInventory and ConsInventory

```
//in class MTInventory
public boolean contains(String toyName) {
   return false;
}
```

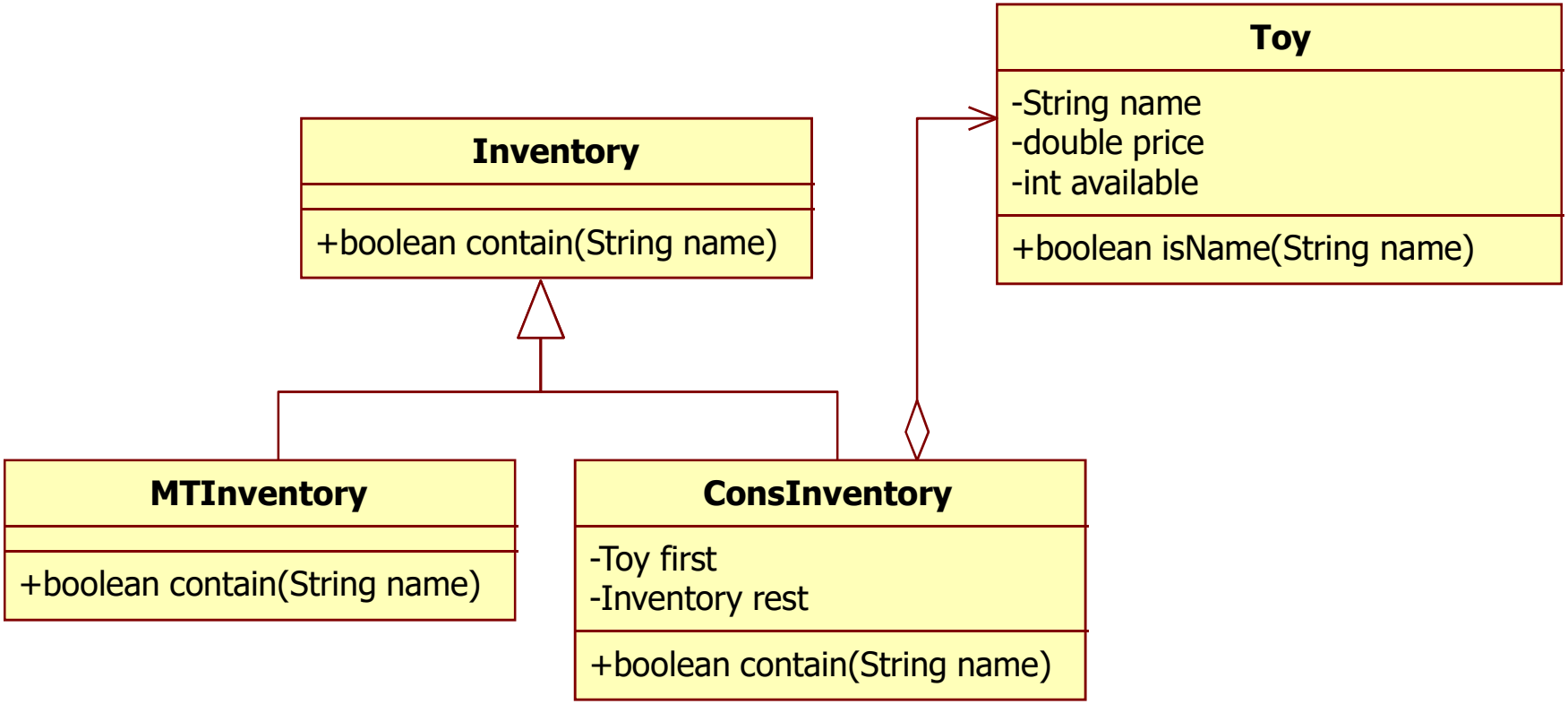
```
// in class ConsInventory
public boolean contains(String toyName) {
   return this.first.isName(toyName)
        || this.rest.contains(toyName);
}
```

```
//in class Toy
 public boolean isName(String toyName) {
      return this.name.equals(toyName);
   }
```

Test contains()

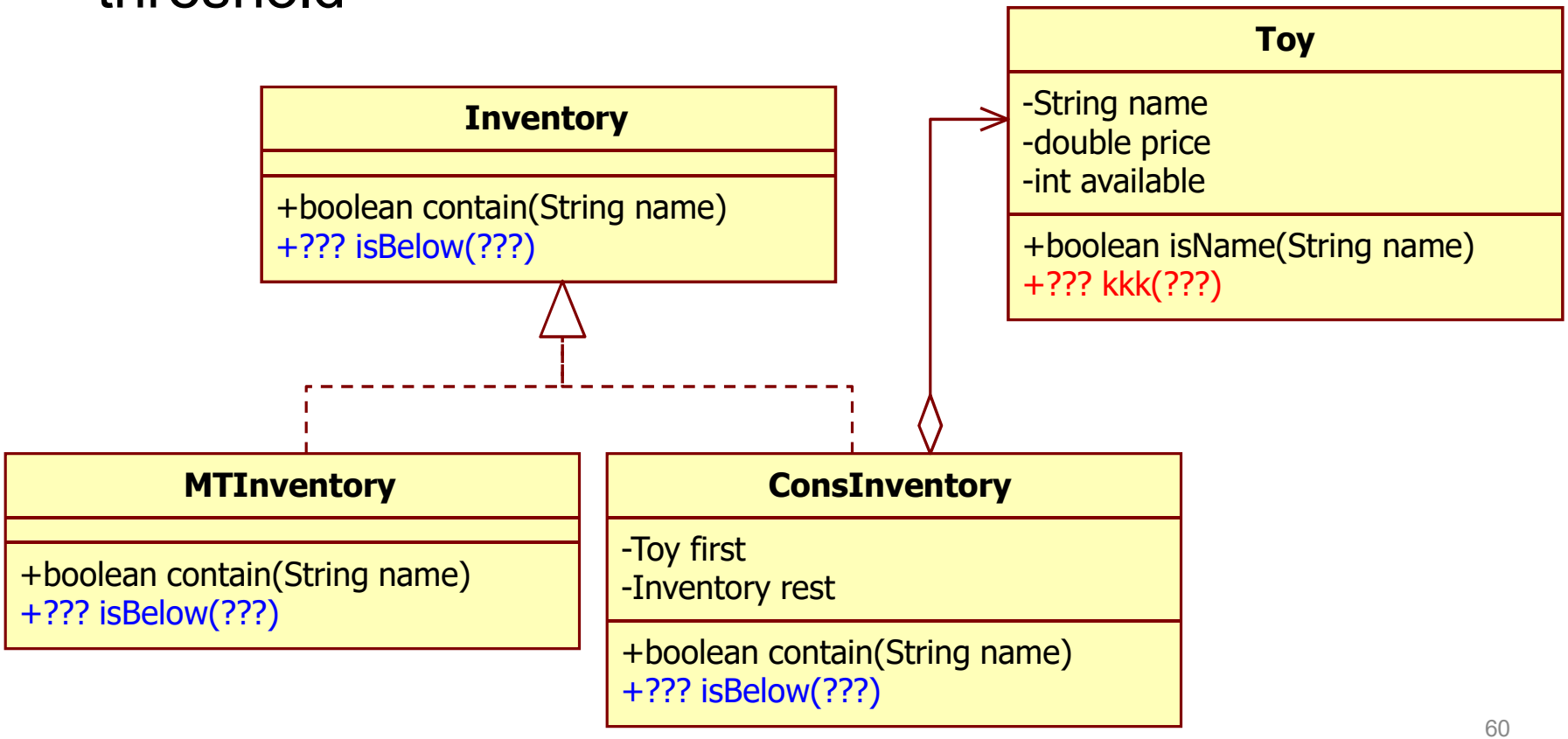
```
public void testContains(){
   Toy doll = new Toy("doll", 17.95, 5);
   Toy robot = new Toy("robot", 22.05, 3);
   Toy gun = new Toy ("gun", 15.0, 4);

   Inventory empty = new MTInventory();
   Inventory i1 = new ConsInventory(doll, empty);
   Inventory i2 = new ConsInventory(robot, i1);
   Inventory all = new ConsInventory(doll,
        new ConsInventory(robot,
        new ConsInventory(gun, new MTInventory())));

   assertFalse(empty.contains("robot"));
   assertfalse(i1.contains("robot"));
   assertTrue(i2.contains("robot"));
   assertTrue(all.contains("robot"));
   assertFalse(all.contains("car"));
}
```

Class diagram after add `contains()`



Add `isBelow` method

- Develop the method isBelow, which checks whether all of the prices of toys in inventory are below the threshold



Purpose and contract of `isBelow()` for `Inventory`

```
public interface Inventory {
   // determines whether or not the name of
   // toy occurs in the Inventory
   public boolean contains(String toyName);

   // determines whether or not all prices of toys
   // in the Inventory below a threshold
   public boolean isBelow(double threshold);
}
```

Examples to test `isBelow()`

```
Toy doll = new Toy("doll", 17.95, 5);
Toy robot = new Toy("robot", 22.05, 3);
Toy gun = new Toy ("gun", 15.0, 4);

Inventory empty = new MTInventory();
Inventory i1 = new ConsInventory(doll, empty);
Inventory i2 = new ConsInventory(robot, i1);
Inventory all = new ConsInventory(doll,
                new ConsInventory(robot,
                new ConsInventory(gun, new MTInventory())));

empty.isBelow(20) → should be true
i1.isBelow(20) → should be true
i2.isBelow(20) → should be false
all.isBelow(20) → should be false
all.isBelow(25) → should be true
```

isBelow() for MTInventory and ConsInventory

```
//inside of MTInventory class
public boolean isBelow(double threshold) {
   return true;
}
```

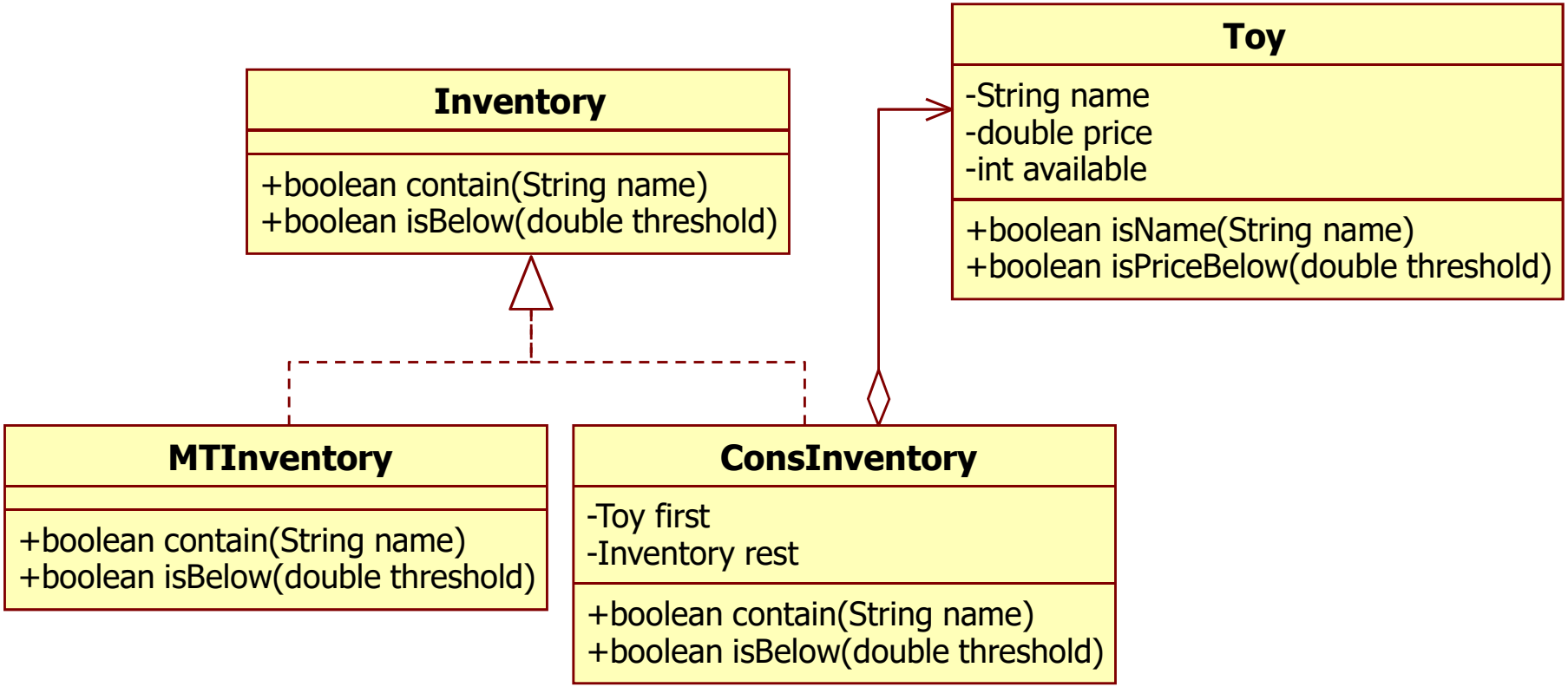
```
// inside of ConsInventory class
public boolean isBelow(double threshold) {
   return this.first.isPriceBelow(threshold)
          && this.rest.isBelow(threshold);
}
```

```
// inside of Toy class
public boolean isPriceBelow(double threshold) {
   return this.price < threshold;
}
```

Test `isBelow()`

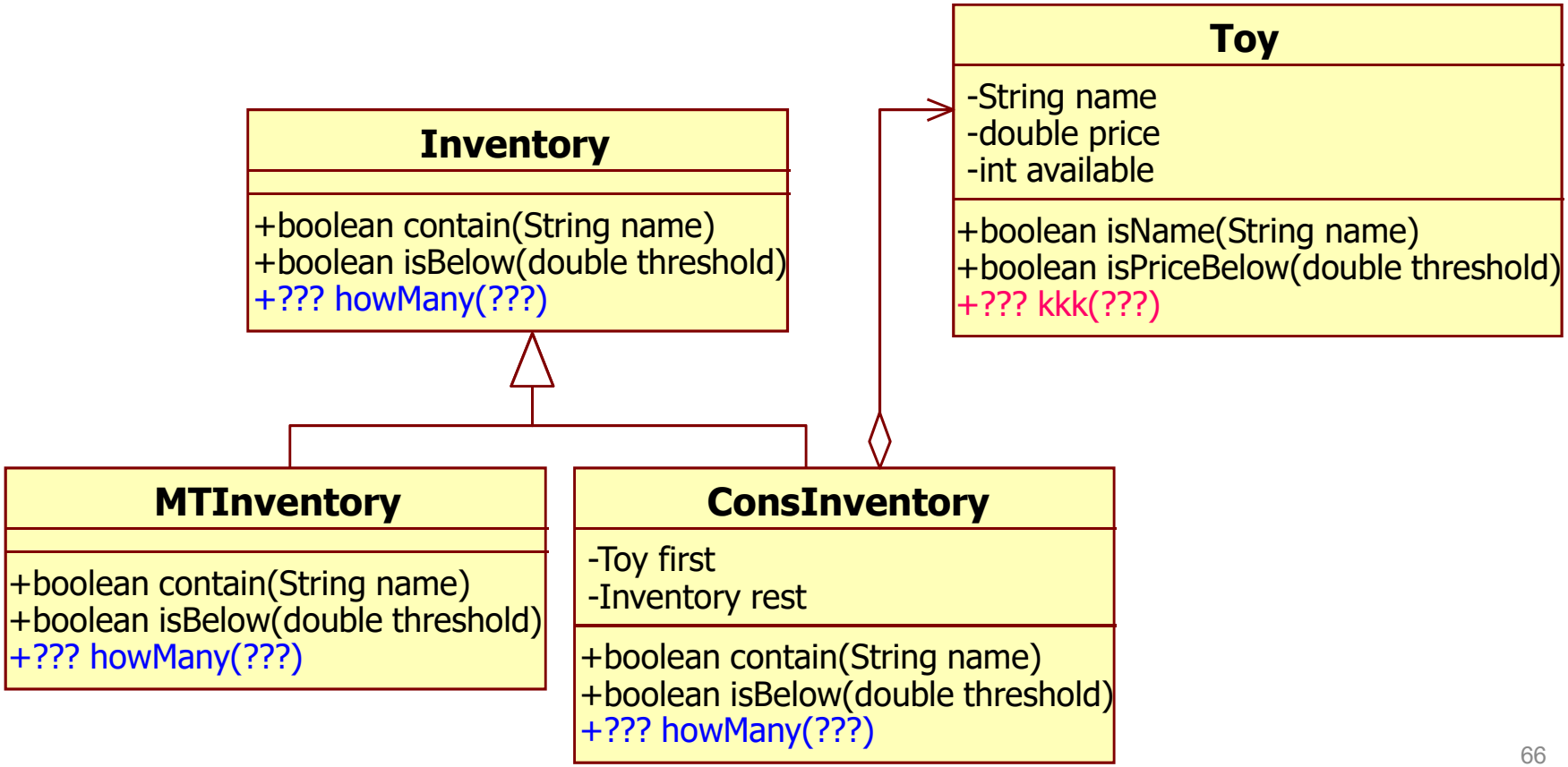
```
public void testIsBellow(){
   Toy doll = new Toy("doll", 17.95, 5);
   Toy robot = new Toy("robot", 22.05, 3);
   Toy gun = new Toy ("gun", 15.0,4);

   Inventory empty = new MTInventory();
   Inventory i1 = new ConsInventory(doll, empty);
   Inventory i2 = new ConsInventory(robot, i1);
   Inventory all = new ConsInventory(doll,
                 new ConsInventory(robot,
                 new ConsInventory(gun, new MTInventory())));

   assertTrue(empty.isbelows(20));
   assertTrue(i1.isBelow(20));
   assertFalse(i2.isBelow(20));
   assertFalse(all.isBelow(20));
   assertTrue(all.isBelow(25));
}
```

Class diagram after add `isBelow()`



Add `howMany` method

- Develop the method `howMany`, which produces the number of toy items in the inventory.



Purpose and contract of `howMany()` for `Inventory`

```
public interface Inventory {
  ...

  // count the number of items in the Inventory
  public int howMany();
}
```

Examples to test howMany()

```
Toy doll = new Toy("doll", 17.95, 5);
Toy robot = new Toy("robot", 22.05, 3);
Toy gun = new Toy ("gun", 15.0, 4);

Inventory empty = new MTInventory();
Inventory i1 = new ConsInventory(doll, empty);
Inventory i2 = new ConsInventory(robot, i1);
Inventory all = new ConsInventory(doll,
                new ConsInventory(robot,
                new ConsInventory(gun, new MTInventory())));

empty.howMany() → should be 0
i1.howMany() → should be 1
i2.howMany() → should be 2
all.howMany() → should be 3
```

howMany() for MTInventory and ConsInventory

```
// inside of MTInventory class
public int howMany() {
   return 0;
}
```

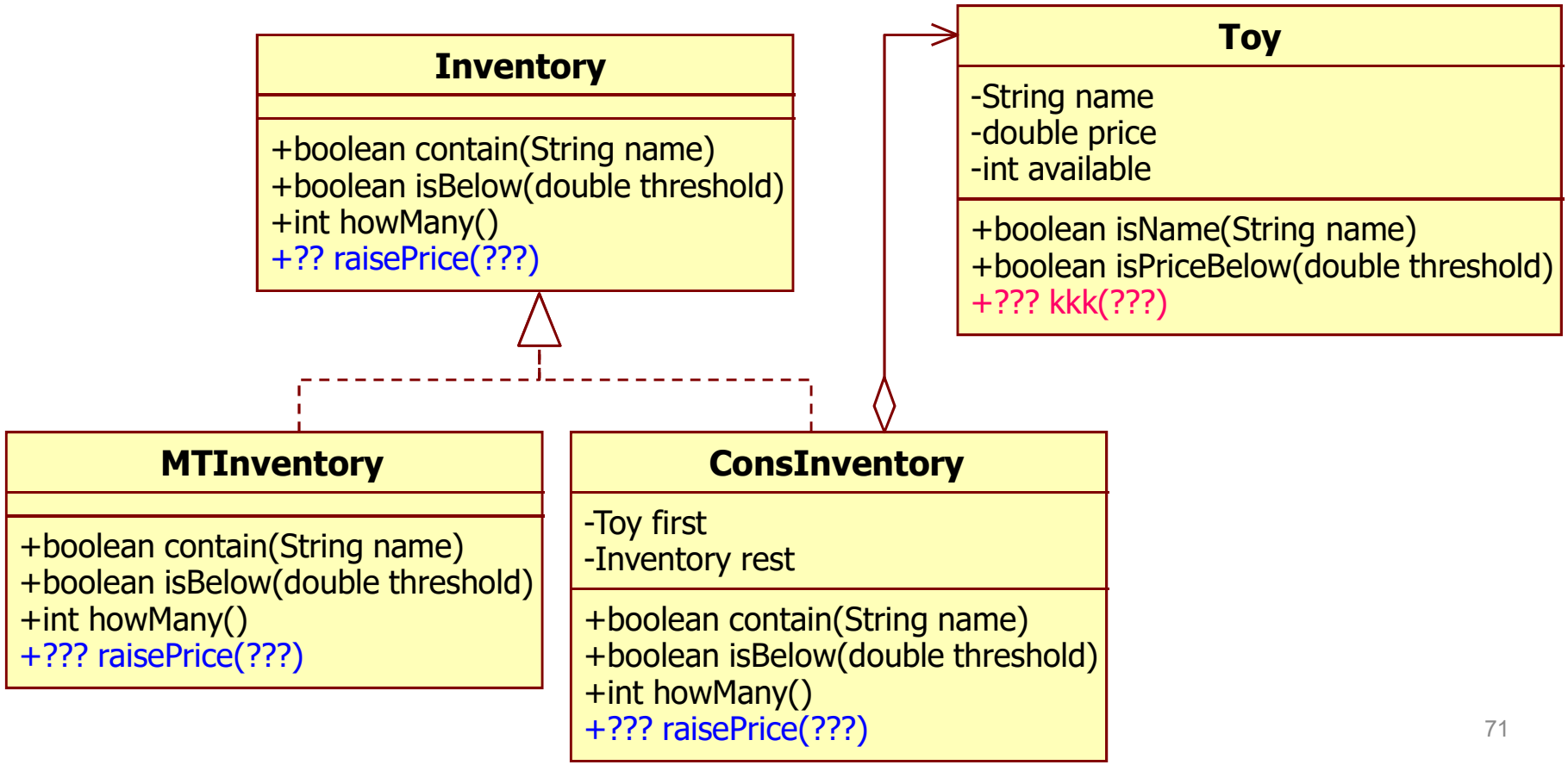
```
// inside of ConsInventory class
public int howMany() {
   return 1 + this.rest.howMany();
}
```

Test howMany()

```
public void testHowMany() {
  Toy doll = new Toy("doll", 17.95, 5);
  Toy robot = new Toy("robot", 22.05, 3);
  Toy gun = new Toy ("gun", 15.0,4);

  Inventory empty = new MTInventory();
  Inventory i1 = new ConsInventory(doll, empty);
  Inventory i2 = new ConsInventory(robot, i1);
  Inventory all = new ConsInventory(doll,
                  new ConsInventory(robot,
                  new ConsInventory(gun, new MTInventory())));

  assertEquals(0, empty.howMany());
  assertEquals(1, i1.howMany());
  assertEquals(2, i2.howMany());
  assertEquals(3, all.howMany());
}
```

Add `raisePrice()` method

- Develop the method `raisePrice`, which produces an inventory in which all prices are raised by a given rate



Purpose and contract of `raisePrice()` for `Inventory` - Immutable version

- **Q**: what does the `raisePrice()` method return?
- **A:** It returns a new `Inventory` whose each element has new price

```
public interface Inventory {
    ...
    // raise all prices of toys
    // in the Inventory with rate
    public Inventory raisePrice(double rate);
}
```

Examples to test `raisePrice()`

```
Toy doll = new Toy("doll", 17.95, 5);
Toy robot = new Toy("robot", 22.05, 3);
Toy gun = new Toy ("gun", 15.0, 4);

Inventory empty = new MTInventory();
Inventory all = new ConsInventory(doll,
                new ConsInventory(robot,
                new ConsInventory(gun, new MTInventory())));

empty.raisePrice(0.05) → should be new MTLog()
all.raisePrice(0.05) → new ConsLog(new Toy("doll", 18.8475, 5),
                  new ConsLog(new Toy("robot", 23.1525, 5),
                  new ConsLog(new Toy("gun", 15.75, 5),
                  new MTLog())))
```

raisePrice() for MTInventory and ConsInventory

```
// inside of MTInventory class
public Inventory raisePrice(double rate) {
   return new MTInventory();
}
```

```
// inside of ConsInventory class
public Inventory raisePrice(double rate) {
   Toy aToy = this.first.copyWithRaisePrice(rate);
   return new ConsInventory(aToy, this.rest.raisePrice(rate));
}
```

```
// inside of Toy class
public Toy copyWithRaisePrice(double rate) {
   return new Toy(this.name,
                  this.price * (1 + rate), this.available);
}
```

Test `raisePrice()`

```
public void testRaisePrice(){
   Toy doll = new Toy("doll", 17.95, 5);
   Toy robot = new Toy("robot", 22.05, 3);
   Toy gun = new Toy ("gun", 15.0,4);

   Inventory all = new ConsInventory(doll,
       new ConsInventory(robot,
       new ConsInventory(gun, new MTInventory())));

   assertEquals(all.raisePrice(0.05),
          new ConsLog(new Toy("doll", 18.8475, 5),
              new ConsLog(new Toy("robot", 23.1525, 5),
                  new ConsLog(new Toy("gun", 15.75, 5),
                      new MTLog()))
               );
   System.out.println(all.raisePrice(0.05));
}
```

equals() method

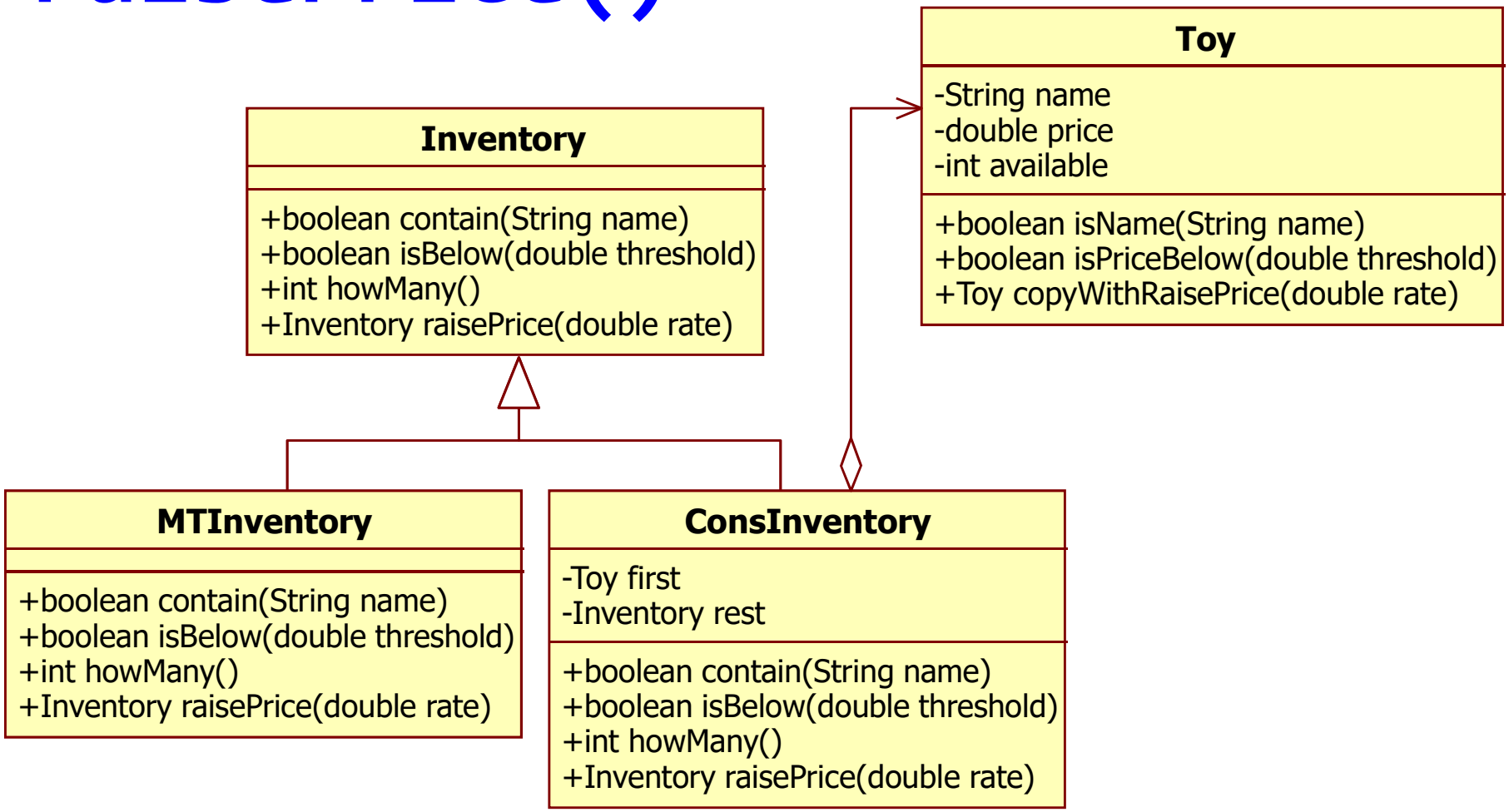
```
// in MTInventory class
public boolean equals(Object obj) {
   if (obj == null || !(obj instanceof MTInventory)
      return false;
   return true;
}
```

```
// inside ConsInventory class
public boolean equals(Object obj) {
      if (obj == null || !(obj instanceof ConsInventory))
         return false;
      else {
         ConsInventory that = (ConsInventory) obj;
         return this.first.equals(that.first)
               && this.rest.equals(that.rest);
      }
   }
```

equals() method in Toy

```
// in Toy class
public boolean equals(Object obj) {
   if (obj == null || !(obj instanceof Toy))
      return false;
   else {
      Toy that = (Toy) obj;
      return this.name.equals(that.name) &&
          this.price == that.price &&
          this.available == that.available;
      }
   }
}
```

Class diagram after add `raisePrice()`



Purpose and contract of `raisePrice()` for `Inventory` - mutable version

- **Q**: what does the `raisePrice()` method return?
- **A:** It just updates `Inventory` whose each element has new price and return `void`.

```
public interface Inventory {
  // raise all prices of toys
  // in the Inventory with rate
  public void raisePriceMutable(double rate);
}
```

raisePrice() for MTInventory and ConsInventory

```
// inside of MTInventory class
public void raisePriceMutable(double rate) { }
```

```
// inside of ConsInventory class
public void raisePriceMutable(double rate) {
   this.first.setNewPrice(rate);
   this.rest.raisePriceMutable(rate);
}
```

Delegation to **Toy** object

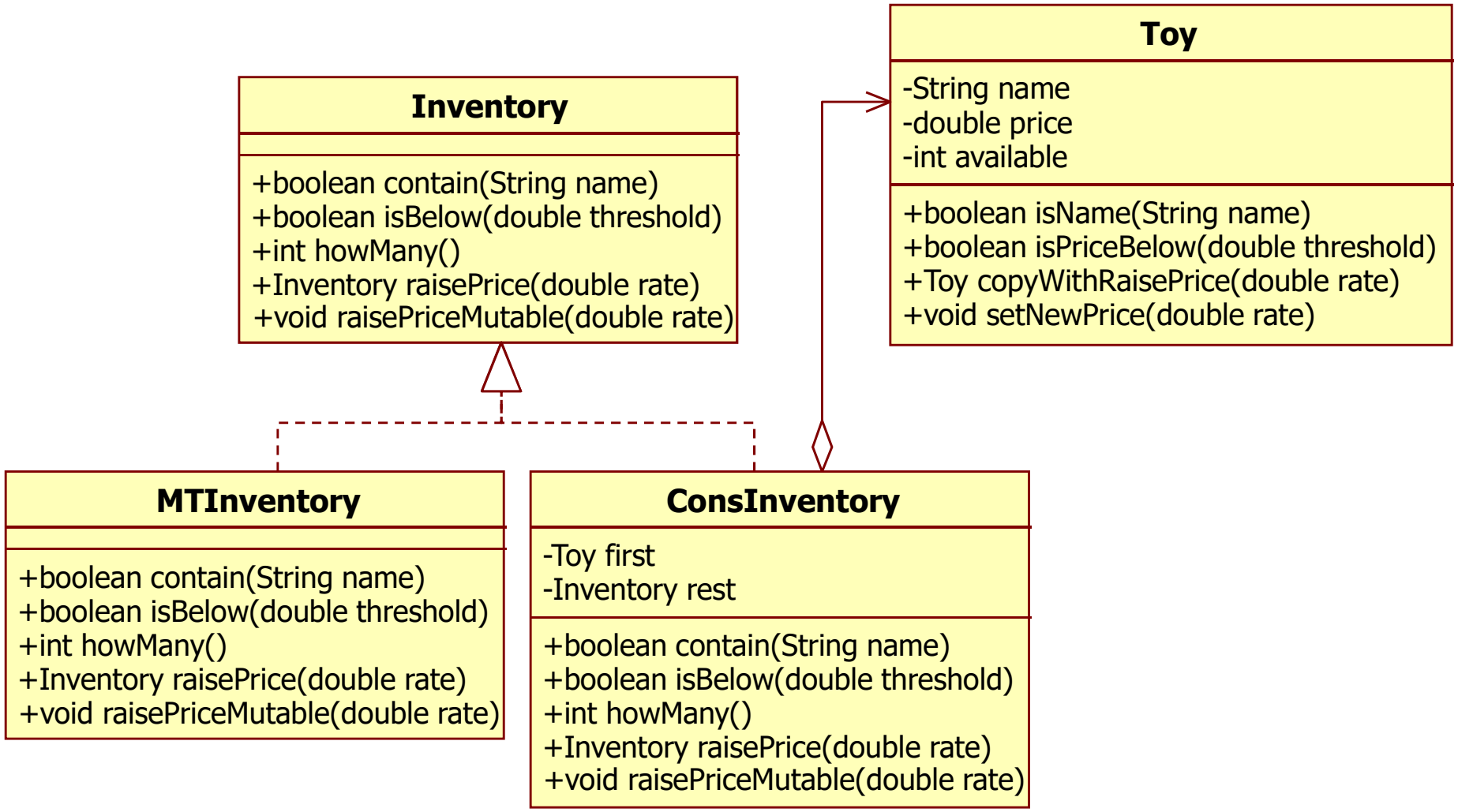
setNewPrice() in **Toy** class

```
public void setNewPrice(double rate) {
   this.price = this.price * (1 + rate);
}
```

Test raisePriceMutable()

```
public void testRaisePrice(){
   Toy doll = new Toy("doll", 17.95, 5);
   Toy robot = new Toy("robot", 22.05, 3);
   Toy gun = new Toy ("gun", 15.0,4);

   Inventory all = new ConsInventory(doll,
       new ConsInventory(robot,
       new ConsInventory(gun, new MTInventory())));

   all.raisePriceMutable(0.05);
   // after invoking raisePriceMutable(rate)
   assertEquals(all,
           new ConsLog(new Toy("doll", 18.8475, 5),
              new ConsLog(new Toy("robot", 23.1525, 5),
                  new ConsLog(new Toy("gun", 15.75, 5),
                      new MTLog()))
               );
   System.out.println(all);
}
```

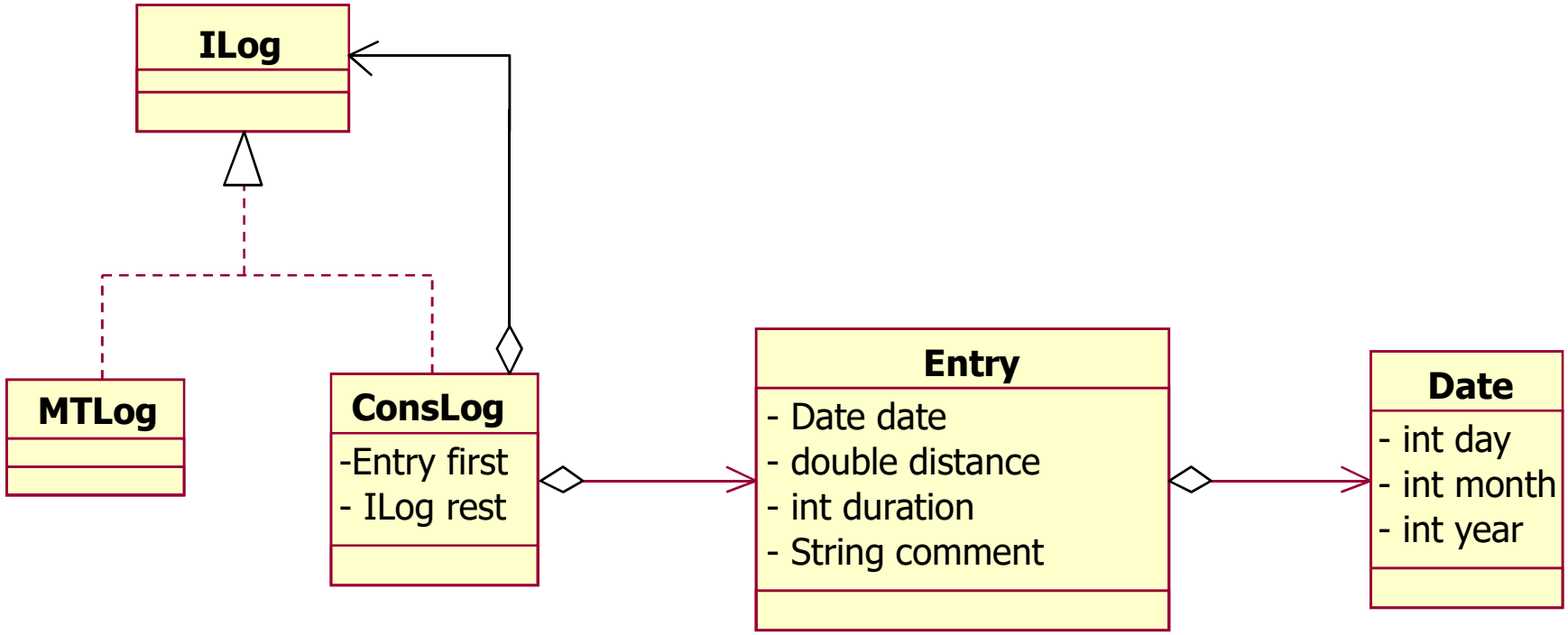
Final class diagram



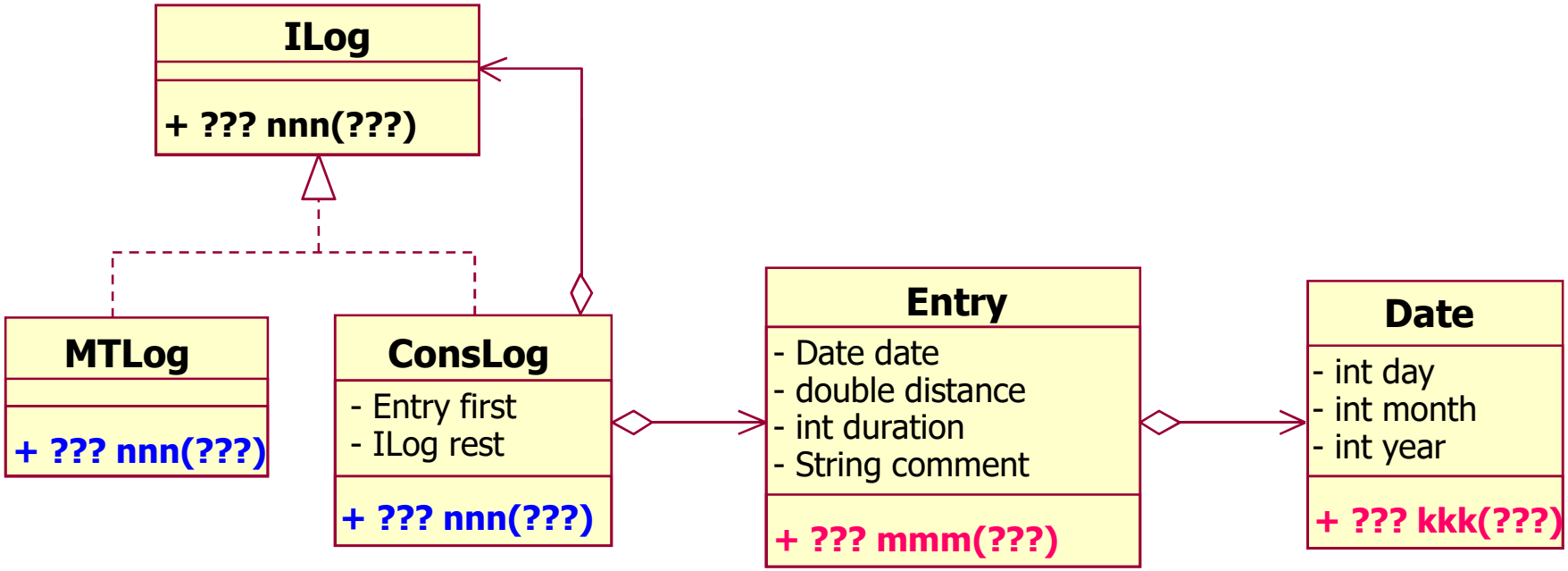
Recall the problem of tracking a runner's workouts

- Develop a program that manages a runner's training log. Every day the runner enters one entry concerning the day's run.
  ... For each entry, the program should compute how fast the runner ran (Exercise 3.1.4 & 3.1.5 in week 1).
  ... The runner may also wish to determine ***the total number of miles run***

Class diagram for a runner's log



Add methods to the runner's log Class Diagram



Q: Write Java method templates for all the classes in the class diagram ?

Java template for `ILog` and `MTLog`

```
public interface ILog {
   public ??? nnn(???);
}
```

```
public class MTLog implements ILog {

  public ??? nnn(???) {
    ...
  }
}
```

Java template for ConsLog

```
public class ConsLog implements ILog {
   private Entry first;
   private ILog rest;

   public ConsLog(Entry first, ILog rest) {
      this.first = first;
      this.rest = rest;
   }

   public ??? nnn(???) {
      ... this.first.mmm(??) ...
      ... this.rest.nnn(??) ...
   }
}
```

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

Java template for `Date`

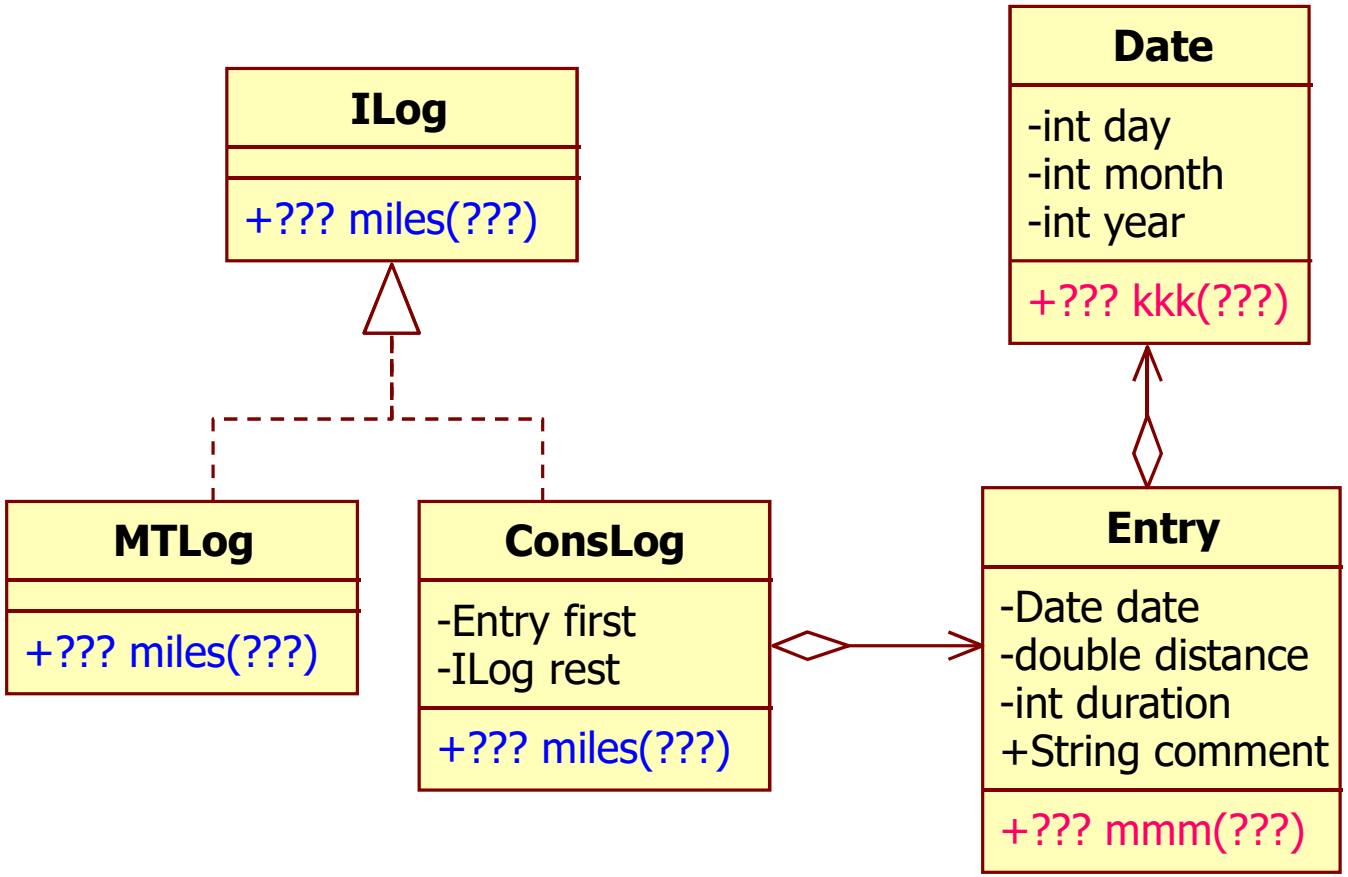
```
public class Date {
   private int day;
   private int month;
   private int year;

   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 ...
   }
}
```

Examples for a runner's log

```
Entry e1 = new Entry(new Date(5, 5, 2005), 5.0, 25, "Good");
Entry e2 = new Entry(new Date(6, 6, 2005), 3.0, 24, "Tired");
Entry e3 = new Entry(new Date(23, 6, 2005), 26.0, 156, "Great");

ILog log = new ConsLog(e1, new ConsLog(e2,
             new ConsLog(e3, new MTLog())));
```

Compute the total number of miles run

- Using the method template for ILog, design a method to compute the total number of miles run



`miles()` for `ILog`

```
public interface ILog {
   // to compute the total number of miles
   // recorded in this log
   public double miles();
}
```

Q: Develop some examples to test the `miles()` method

Examples to test `miles()`

```
Entry e1 = new Entry(new Date(5, 5, 2005), 5.0, 25, "Good");
Entry e2 = new Entry(new Date(6, 6, 2005), 3.0, 24, "Tired");
Entry e3 = new Entry(new Date(23, 6, 2005), 26.0, 156, "Great");

ILog l0 = new MTLog();
ILog l1 = new ConsLog(e1, l0);
ILog l2 = new ConsLog(e2, l1);
ILog l3 = new ConsLog(e3, l2);

l0.miles() → should be 0.0
l1.miles() → should be 5.0
l2.miles() → should be 8.0
l3.miles() → should be 34.0
```

Q: Implement `miles()` in `MTLog` and `ConsLog`

Implement `miles()`

in MTLog

```
public class MTLog implements ILog {
    ...
    public double miles() {
        return 0.0;
    }
}
```

in ConsLog

```
public class ConsLog implements ILog {
    ...
    public double miles() {
        return this.first.getDistance() +
               this.rest.miles();
    }
}
```

getDistance() in Entry

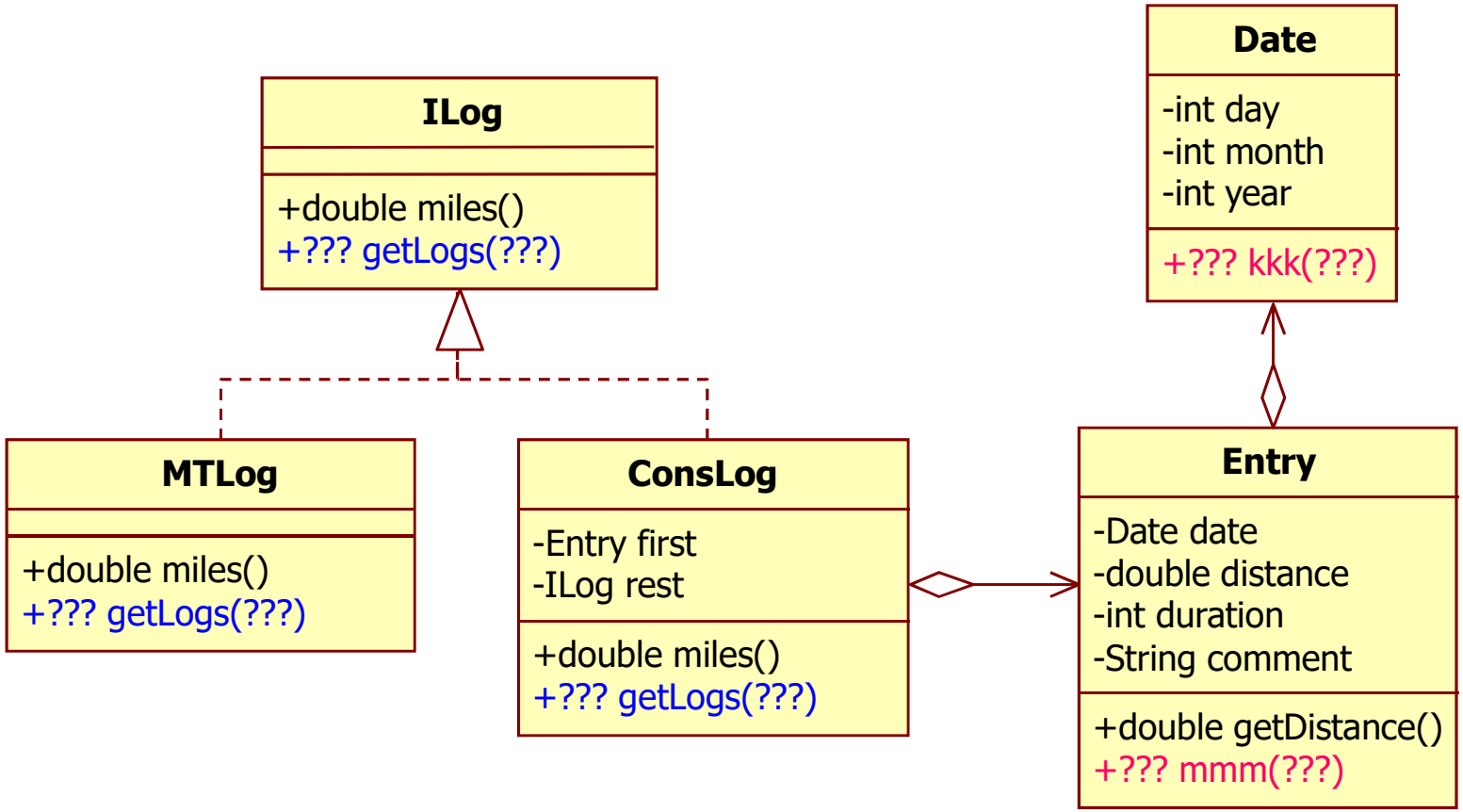
```
public class Entry {
   private Date date;
   private double distance;
   private int duration;
   private String comment;

   ...
   public double getDistance() {
     return this.distance;
   }
}
```

Test `miles()` method

```
public void testMiles() {
   Entry e1 = new Entry(new Date(5, 5, 2005), 5.0, 25, "Good");
   Entry e2 = new Entry(new Date(6, 6, 2005), 3.0, 24, "Tired");
   Entry e3 = new Entry(new Date(23, 6, 2005), 26.0, 156, "Great");

   ILog l0 = new MTLog();
   ILog l1 = new ConsLog(e1, l0);
   ILog l2 = new ConsLog(e2, l1);
   ILog l3 = new ConsLog(e3, l2);

   asserEquals(l0.miles(), 0.0);
   asserEquals(l1.miles(), 5.0);
   asserEquals(l2.miles(), 8.0);
   asserEquals(l3.miles(), 34.0);
}
```

Extension of the runner's log problem

... The runner wants to see ***his log for a specific month*** of his training season. ...



getLogs() for ILog

```
public interface ILog {
   // to compute the total number of miles
   // recorded in this log
   public double miles();

   // to extract those entries in this log
   // for the given month and year
   public ILog getLogs(int month, int year);
}
```

Q: Develop some examples to test the getLogs() method

Examples to test `getLogs()`

```
Entry e1 = new Entry(new Date(5, 5, 2005), 5.0, 25, "Good");
Entry e2 = new Entry(new Date(6, 6, 2005), 3.0, 24, "Tired");
Entry e3 = new Entry(new Date(23, 6, 2005), 26.0, 156, "Great");

ILog l0 = new MTLog();
ILog l1 = new ConsLog(e1, l0);
ILog l2 = new ConsLog(e2, l1);
ILog l3 = new ConsLog(e3, l2);

l0.getLogs(6, 2005) → should be new MTLog()
l1.getLogs(6, 2005) → should be new MTLog()
l2.getLogs(6, 2005) → should be new ConsLog(e2, new MTLog())
l3.getLogs(6, 2005) → should be
   new ConsLog(e3, new ConsLog(e2, new MTLog()))
```

Q: Implement `getLogs()` in `MTLog` and `ConsLog`

getLog() for MTLog

```
public class MTLog implements ILog {
   // ...

   public ILog getLogs(int month, int year) {
      return new MTLog();
   }
}
```

getLogs() for ConsLog

```
public class ConsLog implements ILog {
   private Entry first;
   private ILog  rest;
   // ...

   public ILog getLogs(int month, int year) {
      if (this.first.sameMonthInAYear(month, year))
         return new ConsLog(this.first,
                            this.rest.getLogs(month, year));
      else
         return this.rest.getLogs(month, year);
   }
}
```

sameMonthInAYear() in Entry

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

  // was this entry made in the given month and year
   public boolean sameMonthInAYear(int month, int year) {
      return this.date.sameMonthInAYear(month, year);
   }
}
```

sameMonthInAYear() in 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 boolean sameMonthInAYear(int month, int year) {
      return (this.month == month) &&
             (this.year == year);
   }
}
```

Q: Review delegation ?

Test `getLogs()`

```
public void testGetLogs() {
   Entry e1 = new Entry(new Date(5, 5, 2005), 5.0, 25, "Good");
   Entry e2 = new Entry(new Date(6, 6, 2005), 3.0, 24, "Tired");
   Entry e3 = new Entry(new Date(23, 6, 2005), 26.0, 156, "Great");

   ILog l0 = new MTLog();
   ILog l1 = new ConsLog(e1, l0);
   ILog l2 = new ConsLog(e2, l1);
   ILog l3 = new ConsLog(e3, l2);

   assertEquals(l0.getLogs(6, 2005), new MTLog());
   assertEquals(l1.getLogs(6, 2005), new MTLog());
   assertEquals(l2.getLogs(6, 2005),
                new ConsLog(e2, new MTLog()));
   assertEquals(l3.getLogs(6, 2005),
                new ConsLog(e3, new ConsLog(e2, new MTLog())));
}
```

equals() method

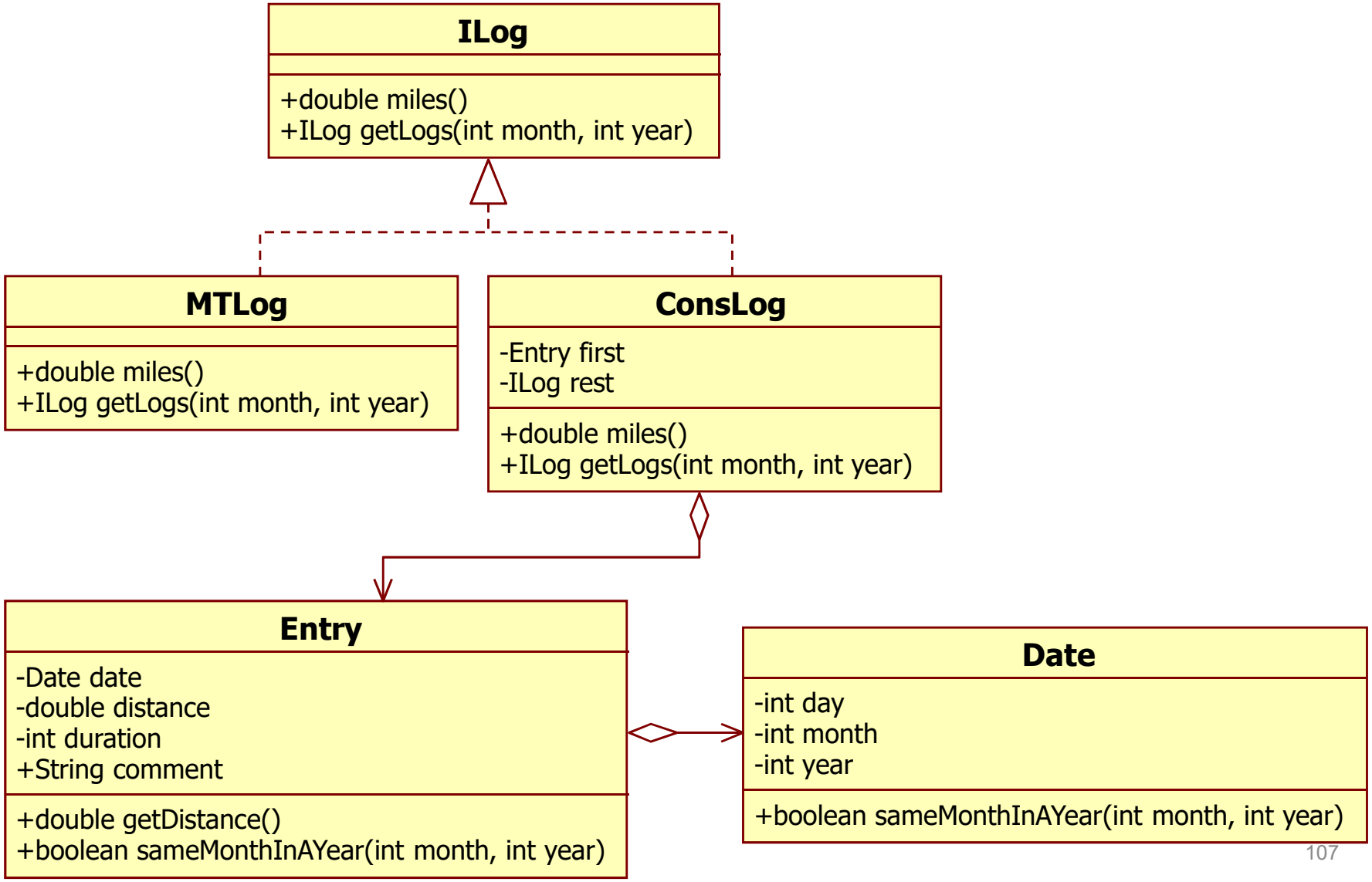
```
// in MTLog class
public boolean equals(Object obj) {
   if (obj == null || !(obj instanceof MTLog)
      return false;
   return true;
}
```

```
// inside ConsLog class
public boolean equals(Object obj) {
      if (obj == null || !(obj instanceof ConsLog))
         return false;
      else {
         ConsLog that = (ConsLog) obj;
         return this.first.equals(that.first)
               && this.rest.equals(that.rest);
      }
   }
```

```java
// in Entry class
public boolean equals(Object obj) {
   if (obj == null || !(obj instanceof Entry))
      return false;
   else {
      Entry that = (Entry) obj;
      return this.date.equals(that.date) &&
          this.distance == that.distance &&
          this.durationInMinutes == that.durationInMinutes &&
          this.postRunFeeling.equals(that.postRunFeeling);
      }
   }
}
```

```java
// inside Date class
public boolean equals(Object obj) {
   if (obj == null || !(obj instanceof Date))
      return false;
   else {
      Date that = (Date) obj;
      return this.day == that.day &&
             this.month == that.month &&
             this.year == that.year;
   }
}
```

Class diagram



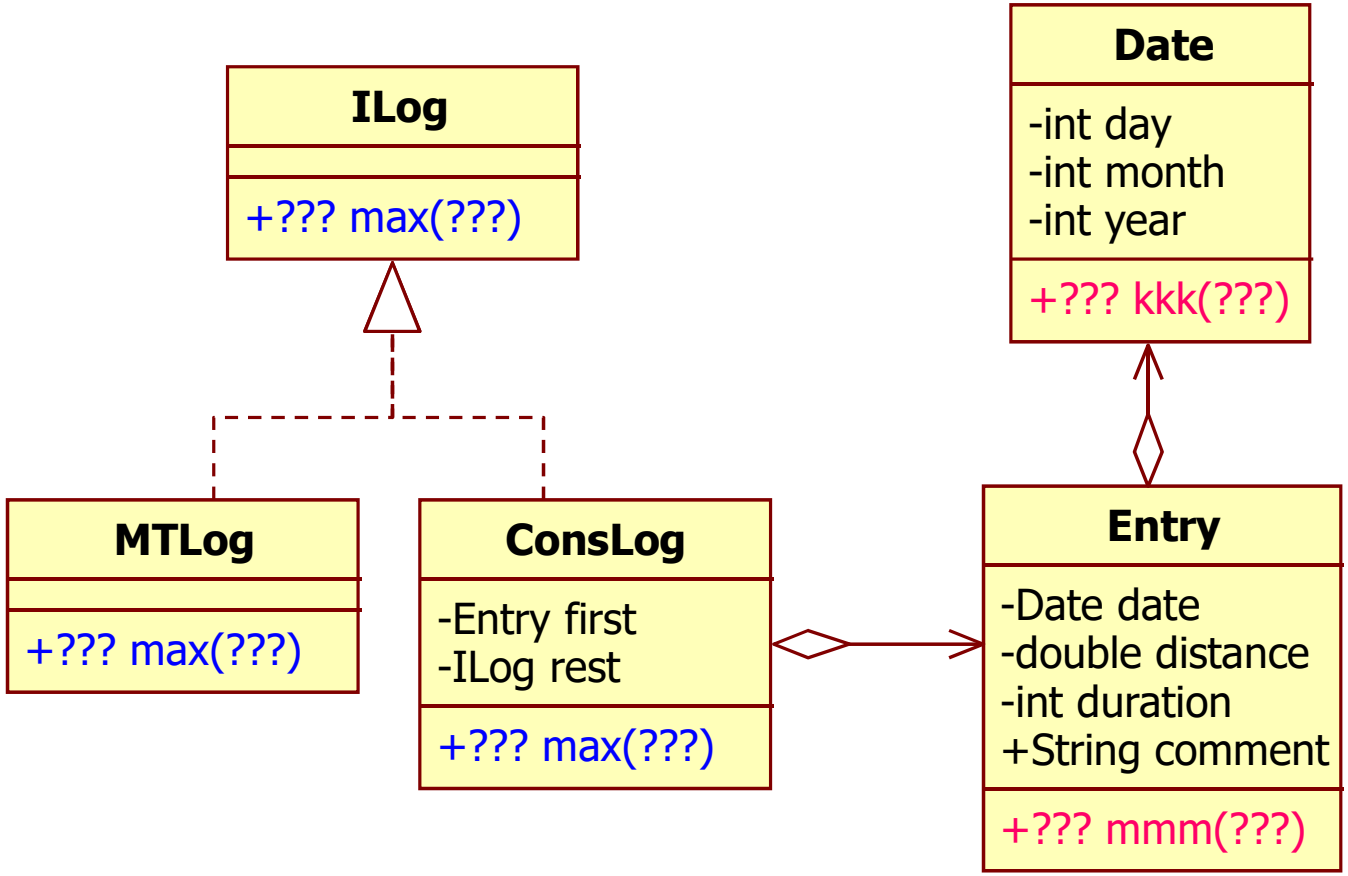
More requirements

Suppose the requirements for the program that tracks a runner's log includes this request:

1. The runner wants to know **the total distance run in a given month**...
   Design the method that computes this number
2. A runner wishes to know **the maximum distance ever run** ... Design the method that computes this number. Assume that the method produces 0 if the log is empty.

Extension: Find **the entry with the largest running distance**

Compute the maximum distance



max() for ILog

```
public interface ILog {
   // to compute the maximum distance
   // in this log
   public double max();
}
```

Examples to test `max()`

```
Entry e1 = new Entry(new Date(5, 5, 2005), 5.0, 25, "Good");
Entry e2 = new Entry(new Date(6, 6, 2005), 3.0, 24, "Tired");
Entry e3 = new Entry(new Date(23, 6, 2005), 26.0, 156, "Great");

ILog l0 = new MTLog();
ILog l1 = new ConsLog(e1, l0);
ILog l2 = new ConsLog(e2, l1);
ILog l3 = new ConsLog(e3, l2);

l0.max() → should be 0.0
l1.max() → should be 5.0
l2.max() → should be 5.0
l3.max() → should be 26.0
```

Q: Implement `miles()` in `MTLog` and `ConsLog`

Implement `max()`

in MTLog

```
public class MTLog implements ILog {
    ...
    public double max() {
        return 0.0;
    }
}
```

in ConsLog

```
public class ConsLog implements ILog {
    ...
    public double max() {
        return Math.max(this.first.getDistance(),
                this.rest.max());
    }
}
```

Test `max()` method

```
public void testMax() {
   Entry e1 = new Entry(new Date(5, 5, 2005), 5.0, 25, "Good");
   Entry e2 = new Entry(new Date(6, 6, 2005), 3.0, 24, "Tired");
   Entry e3 = new Entry(new Date(23, 6, 2005), 26.0, 156, "Great");

   ILog l0 = new MTLog();
   ILog l1 = new ConsLog(e1, l0);
   ILog l2 = new ConsLog(e2, l1);
   ILog l3 = new ConsLog(e3, l2);

   asserEquals(l0.max(), 0.0);
   asserEquals(l1.max(), 5.0);
   asserEquals(l2.max(), 5.0);
   asserEquals(l3.max(), 26.0);
}
```

Design `max()` for `ILog` (version 2)

```
public interface ILog {
    ...
    // to compute the maximum distance
    // in this log with the current max
    public double max(double current);
}
```

Implement `max()`

in MTLog

```
public class MTLog implements ILog {
    ...
    public double max(double current) {
        return current;
    }
}
```

in ConsLog

```
public class ConsLog implements ILog {
    ...
    public double max(double current) {
        if (this.first.getDistance() > current)
            return this.rest.max(this.first.getDistance());
        else return this.rest.max(current);
    }
}
```

Test `max()` method

```
public void testMiles() {
   Entry e1 = new Entry(new Date(5, 5, 2005), 5.0, 25, "Good");
   Entry e2 = new Entry(new Date(6, 6, 2005), 3.0, 24, "Tired");
   Entry e3 = new Entry(new Date(23, 6, 2005), 26.0, 156, "Great");

   ILog l0 = new MTLog();
   ILog l1 = new ConsLog(e1, l0);
   ILog l2 = new ConsLog(e2, l1);
   ILog l3 = new ConsLog(e3, l2);

   asserEquals(l0.max(0.0), 0.0);
   asserEquals(l1.max(0.0), 5.0);
   asserEquals(l2.max(0.0), 5.0);
   asserEquals(l3.max(0.0), 26.0);
}
```

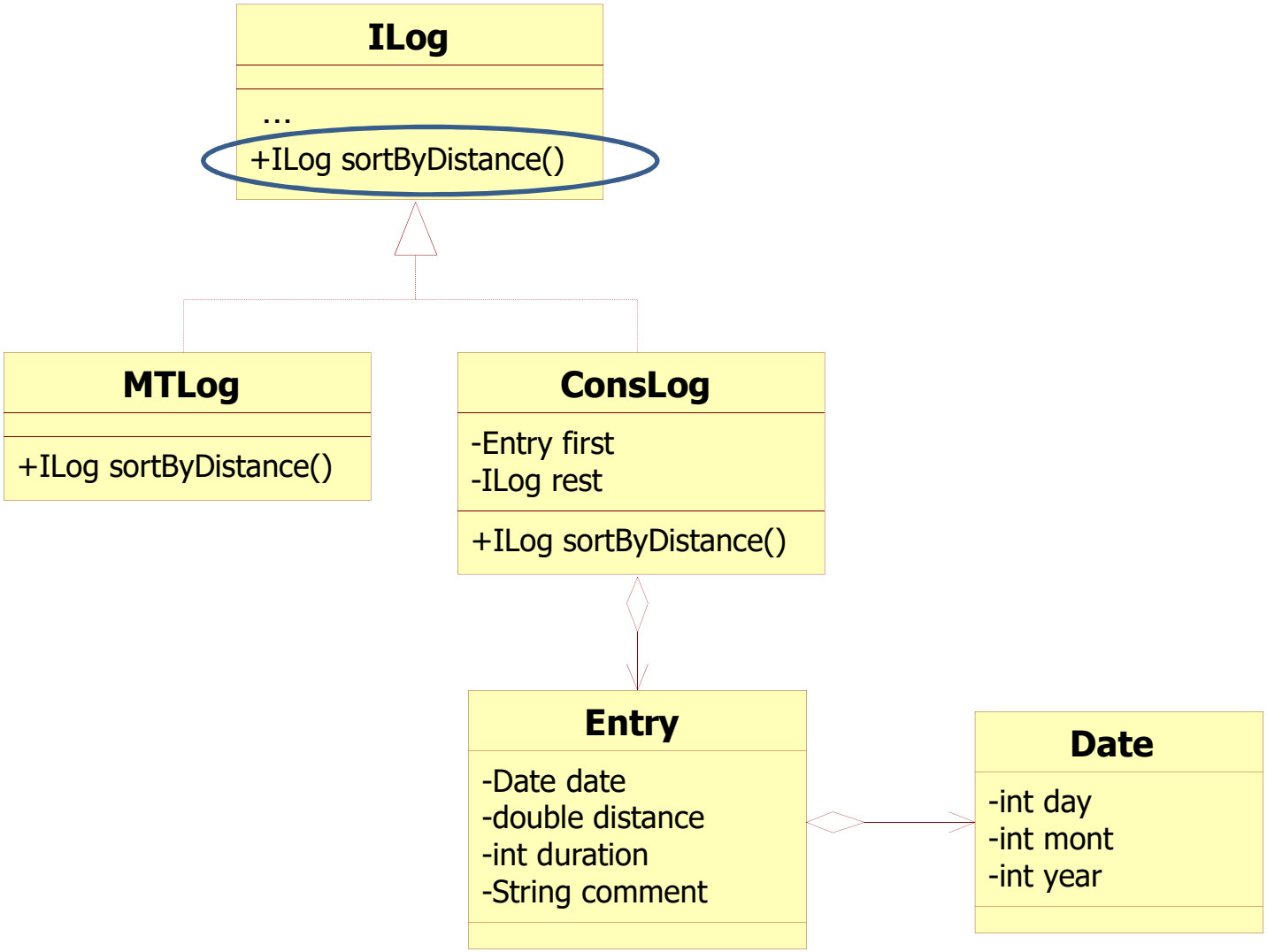
List Sorting

Problem Statement

The runner would like to see the log with entries ordered according to the distance covered in each run, from the shortest to the longest distance.

- **Q:** Which class should this operation belong to?

Modification of `ILog`



Examples

```
Entry e1 = new Entry(new Date(5, 5, 2005), 5.0, 25, "Good");
Entry e2 = new Entry(new Date(6, 6, 2005), 3.0, 24, "Tired");
Entry e3 = new Entry(new Date(23, 6, 2005), 26.0, 156, "Great");

ILog l0 = new MTLog();
ILog l1 = new ConsLog(e1, l0);
ILog l2 = new ConsLog(e2, l1);
ILog l3 = new ConsLog(e3, l2);

l0.sortByDistance() → should be new MTLog()
l1.sortByDistance() → should be new ConsLog(e1, new MTLog())

l2.sortByDistance()
→ should be new ConsLog(e2, new ConsLog(e1, new MTLog()))

l3.sortByDistance()
→ should be new ConsLog(e2, new ConsLog(e1,
                                new ConsLog(e3, new MTLog()))
```

sortByDistance() in ILog

```
public interface ILog {
   // ...

   // to create from this log a new log with
   // entries sorted by distance
   public ILog sortByDistance();
}
```

sortByDistance() in MTLog

```
public class MTLog implements ILog {
  public MTLog() { }
  // ...

  public ILog sortByDistance() {
    return new MTLog();
  }
}
```

Template of `sortByDistance()` in `ConsLog`

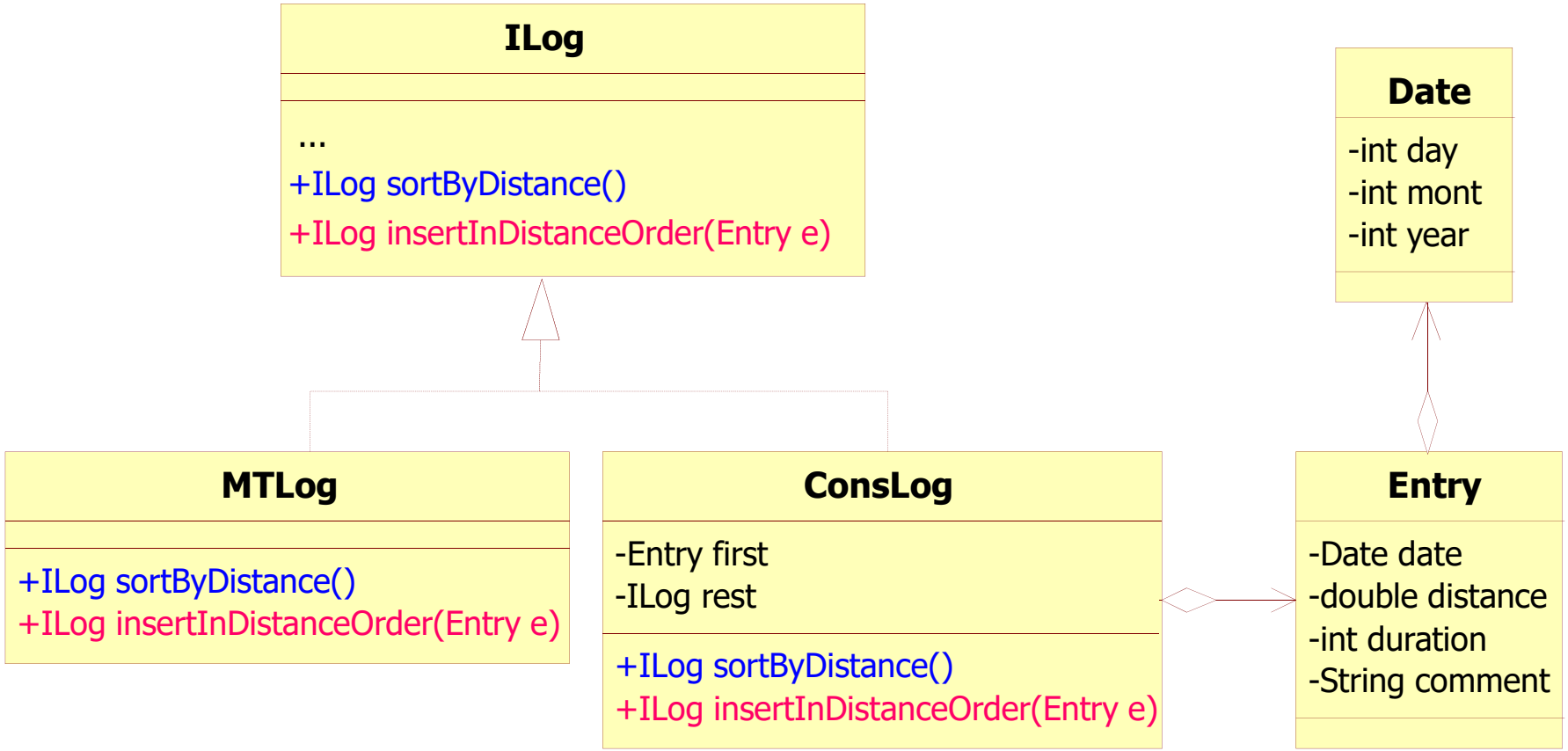
```
public class ConsLog implements ILog {
   private Entry first;
   private ILog rest;
   // ...

   public ILog sortByDistance() {
      ... this.first.mmm(??) ...
      ... this.rest.sortByDistance() ...
   }
}
```

Solution of `sortByDistance()` in `ConsLog`

- To sort a `ConsLog`, we need to insert the `first` entry into the **sorted version** of `rest` to obtained the whole sorted log.

```
public class ConsLog implements ILog {
   private Entry first;
   private ILog rest;
   //...

   public ILog sortByDistance() {
      return this.rest.sortByDistance()
                  .insertInDistanceOrder(this.first);
   }
}
```

Modification of `ILog`



insertInDistanceOrder() in ILog

```
public interface ILog {
  // ...
  // to create from this log a new log with
  // entries sorted by distance
  public ILog sortByDistance();

  // insert the given entry into
  // this sorted log
  public ILog insertInDistanceOrder(Entry e);
}
```

Examples for `insertInDistanceOrder()`

```
Entry e1 = new Entry(new Date(5, 5, 2005), 5.0,  25, "Good");
Entry e2 = new Entry(new Date(6, 6, 2005), 3.0,  24, "Tired");
Entry e3 = new Entry(new Date(23, 6, 2005), 26.0, 156, "Great");
Entry e4 = new Entry(new Date(15, 7, 2005), 10.0, 61, "Tierd");

ILog l0 = new MTLog();
ILog l1 = l0.insertInDistanceOrder(e1);
// should be new new ConsLog(e1, new MTLog()))

ILog l2 = l1.insertInDistanceOrder(e2);
// should be new ConsLog(e2, new ConsLog(e1, new MTLog()))

ILog l3 = l2.insertInDistanceOrder(e3);
// should be  new ConsLog(e2, new ConsLog(e1,
                new ConsLog(e3, new MTLog())))

ILog l4 = l3.insertInDistanceOrder(e4);
// should be new ConsLog(e2, new ConsLog(e1, new ConsLog(e4,
                new ConsLog(e3, new MTLog())))
```

insertInDistanceOrder() in MTLog

```
public class MTLog implements ILog {
   public MTLog() { }
   // ...

   public ILog sortByDistance() {
      return new MTLog();
   }

   public ILog insertInDistanceOrder(Entry e) {
      return new ConsLog(e, new MTLog());
   }
}
```

insertInDistanceOrder() in ConsLog

```
public class ConsLog implements ILog {
   private Entry first;
   private ILog rest;

   public ILog sortByDistance() {
     return this.rest.sortByDistance()
             .insertInDistanceOrder(this.first);
   }

   public ILog insertInDistanceOrder(Entry e) {
      if (e.hasDistanceShorterThan(this.first))
         return new ConsLog(e, this);
      else
         return new ConsLog(this.first,
               this.rest.insertInDistanceOrder(e));
  }
}
```

hasDistanceShorterThan() in Entry

```
public class Entry {
   private Date date;
   private double distance;
   private int duration;
   private String comment;
   // ...

   public boolean hasDistanceShorterThan(Entry that) {
      return this.distance < that.distance;
   }
}
```

Test sortByDistance()

```
public void testSortByDistance() {
   Entry e1 = new Entry(new Date(5, 5, 2005), 5.0, 25, "Good");
   Entry e2 = new Entry(new Date(6, 6, 2005), 3.0, 24, "Tired");
   Entry e3 = new Entry(new Date(23, 6, 2005), 26.0, 156, "Great");

   ILog l0 = new MTLog();
   ILog l1 = new ConsLog(e1, l0);
   ILog l2 = new ConsLog(e2, l1);
   ILog l3 = new ConsLog(e3, l2);

   assertEquals(l0.sortByDistance(), new MTLog());
   assertEquals(l1.sortByDistance(), new ConsLog(e1, MTLog()));
   assertEquals(l2.sortByDistance(),
            new ConsLog(e2, new ConsLog(e1, new MTLog())));
   assertEquals(l3.sortByDistance(), new ConsLog(e2,
            new ConsLog(e1, new ConsLog(e3, new MTLog()))));
}
```

Relax &

…Do Exercises …

Too much hard exercises now

Try again, never stop practicing!