Cats and Roses are two kinds of GardenThing. A Garden contains GardenThings, but is not itself a GardenThing. Cat extends Mammal, and Rose extends Plant. GardenThings accept GardenV visitors (e.g. Rain, Sunshine, and Status). Rain makes a Cat unhappy; Sunshine makes it happy. After a Rose is visited by Rain and Sunshine (in either order) it is blooming. A Status prints out "purr" if a Cat is happy, and "hiss" if it is unhappy. If a Status visits a Rose that is blooming, it prints "Blooming!!!" otherwise it does not print anything.

```java
Cat cat = new Cat("Fritz");
Rose rose = new Rose();
Garden garden = new Garden();

garden.enter(c); // the cat enters the garden
garden.plant(r); // the rose is planted in the garden
```

The following loops through the seven days of the week. Each day the garden is visited by either Sunshine or Rain, followed by the Status visitor. Note that garden.acceptVisitor passes the visitor to all the Cats and Roses in the Garden. Presume that the method goodWeather returns a boolean.

```java
Status status = new Status();
for (int i = 0; i < 7; i++) {
    GardenV v;
    if (goodWeather())
        v = new Sunshine();
    else
        v = new Rain();
    garden.acceptVisitor(v);
    garden.acceptVisitor(status);
}
```

Depending on what goodWeather returns, one result might be "purr" after the first day, then "hiss" and "Blooming!!!" the second day, etc.

1) [15%] **Indicate** any errors and **explain** why they are errors.

a) GardenThing gt = new GardenThing();

b) GardenThing gt = new Cat("Fritz");
   ((Cat)gt).getName();

c) GardenThing gt = new Garden();
   gt.acceptVisitor(new Sunshine());

2) [20%] You are given Mammal (which is not a GardenThing):

```java
abstract public class Mammal extends Vertebrate {
    private String name;

    public Mammal(String s) { name = s; }
    public String getName() { return name; }
}
```

// first write GardenThing here:

```java
// GardenThing
Now finish writing Cat and Rose:

public class Cat extends Mammal implements Item,
    private boolean happiness;

public Cat(String s) {

    public void   setHappy()   { happiness = true; }
    public void   setUnhappy() { happiness = false; }
    public boolean happy()     { return happiness; }

} // Cat

public class Rose extends Plant implements Item,
    private boolean gotRain;
    private boolean gotSun;

public void    receiveRain()     { gotRain = true; }
public void    receiveSunshine() { gotSun  = true; }
public boolean blooming()        { return gotRain && gotSun; }
```

} // Rose

3) [20%] On a back page, write the code of the visitor hierarchy (GardenV, Rain, Sunshine and Status).
4 [15%] Recall the CollectionI interface:

```java
public interface CollectionI {
    boolean add(Item i);
    Item get();
    void start();
    boolean more();
    ... others removed so you don't get tempted to meander off into trouble
}
```

Note that as Cats enter and Roses are planted they are added to contents (a CollectionI). When a Garden accepts a GardenV it passes that visitor to each item in contents. **Write Garden's acceptVisitor method.**

```java
public class Garden {
    private CollectionI contents;

    public Garden() { contents = new ListC(); }

    public void enter(Cat c) { contents.add(c); }
    public void plant(Plant p) { contents.add(p); }

    public void acceptVisitor(
```
5) [10%] On a back page draw a simple **UML class diagram** for **Garden**, **GardenThing**, **Rose**, **Cat**, **Mammal** and **Plant**. Only for **Cat**, include *all* instance variables, methods and annotate with access and type. Label "has a" lines with associated variable names.

6) [10%] On a back page create a complete **UML class diagram** for your visitor hierarchy (**GardenV**, **Rain**, **Sunshine**, and **Status**).

7) [10%] Create a **UML sequence diagram** for the following lines in Driver:

   ```java
   Cat c = new Cat("Fritz");
   Rain r = new Rain();
   c.acceptVisitor(r);
   ```

   Be sure to show *all* method calls between Driver, c, and r. Label all arrows and include all return arrows. *Be sure to include the calls that r makes during the visit with C.*

   Driver           c:Cat           r:Rain