University of British Columbia
CPSC 111, Intro to Computation
2009W2: Jan-Apr 2010

Tamara Munzner

More Class Design

Lecture 30, Mon Mar 29 2010

borrowing from slides by Kurt Eiselt

http://www.cs.ubc.ca/~tmm/courses/111-10

News

- midterm 2 exam papers handed back today
  - raw and scaled scores were made visible Fri
- to reach me, send real email to tmm@cs.ubc.ca
  - please do NOT use the WebCT/Vista Mail
- A3 will be out Wed since A2 due date extension to Tue

News: Labs Reminder

- week 10 is standard lab
- week 11 optional midterm review/correction
  - solutions handed out at end of week 12 labs
- week 12 is standard lab

News: Email

- To reach me, send real email to tmm@cs.ubc.ca
    - please do NOT use the WebCT/Vista Mail

Reading

- Weeklies due either this Wed 3/31 or next Wed 4/7 (since no class Fri, Mon)

- This week:
  - 8.1-.9 (3rd ed)
  - 9.1-9.9 (2nd ed)

Recap: Interfaces as Contract

- Can write code that works on anything that fulfills contract
  - even classes that don’t exist yet!
- Example: Comparable
  - useful if you need to sort items
  - **`compareTo(object)`**
    - returns int < 0 if this object less than parameter
    - returns 0 if same
    - returns int > 0 if this object greater than parameter

Recap: Comparable

- sort method that works on array of objects of any type that implements `Comparable`
  - type guaranteed to have `compareTo` method

- we need to sort
  - `Bunny`
  - `Giraffe`
  - `String`
  - `...`

Recap: Wrappers

- Many classes implement Comparable interface
  - Byte, Character, Double, Float, Integer, Long, Short, String
  - each implements own version of compareTo
- Wrapper classes
  - wraps up (encapsulates) primitive type
  - Double: object wrapping primitive double
    - No: **`sort( double[] myData );`**
    - Yes: **`sort( Double[] myData );`**

Recap: Multiple Interfaces

- Classes can implement more than one interface at once
  - contract to implement all abstract methods defined in every interface it implements

```
public class MyClass implements Interface1, Interface2,
   Interface3
{
}
```

Recap: Selection Sort For Int Primitives

```
// selection sort
public class SortTest1
{
  public static void main(String[] args)
  {
    int[] numbers = {16,3,19,8,12};
    int min, temp;
    //select location of next sorted value
    for (int i = 0; i < numbers.length-1; i++)
    {
      min = i;
      //find the smallest value in the remainder of
      //the array to be sorted
      for (int j = i+1; j < numbers.length; j++)
      {
        if (numbers[j] < numbers[min])
        {
          min = j;
        }
      }
      //swap two values in the array
      temp = numbers[i];
      numbers[i] = numbers[min];
      numbers[min] = temp;
    }

    System.out.println("Printing sorted result");
    for (int i = 0; i < numbers.length; i++)
    {
      System.out.println(numbers[i]);
    }
  }
}
```

Finishing Comparable Code

Bunny Class Warmup

Question 4: **[15 marks]**

Now let's use Java to simulate bunnies! (Why? Because everybody likes bunnies!) In our simulation, each bunny is on a grid at some location defined by an X-coordinate and a Y-coordinate. Also, each bunny has some number of energy units measured in carrot sticks. (X-coordinates, Y-coordinates, and the number of carrot sticks are integer values.) Bunnies can hop north, south, east, or west. When a bunny hops to the north, the bunny's Y-coordinate is increased by 1, and the X-coordinate remains unchanged. When a bunny hops to the west, the bunny's X-coordinate is decreased by 1, and the Y-coordinate remains unchanged. Same idea for hops east (X-coordinate increased by 1, Y-coordinate unchanged) and south (Y-coordinate decreased by 1, X-coordinate unchanged). Note that making one hop requires a bunny to eat one carrot stick, and when a bunny has eaten all of his or her carrot sticks, that bunny can't hop.

Use Java to create a Bunny class which can be used to generate Bunny objects that behave as described above. When a new Bunny object is created, the Bunny always starts at coordinates X = 10, Y = 10, and the Bunny has 5 carrot sticks. Your Bunny class definition must include a hop(int direction) method, and a displayInfo() method. The direction parameter is 12 for north, 3 for east, 6 for south, and 9 for west (like a clock face). The hop() method should test to make sure that the Bunny has not eaten all the carrot sticks – if the Bunny still has carrot sticks, the hop() method should update coordinates as explained above and print the message "hop". If no carrot sticks remain, it should just print the message "This bunny can't hop".
The displayInfo() method should print the Bunny's location and number of remaining carrot sticks. Below is a simple test program that could be used to test your Bunny class definition, followed by the output we'd expect to see when using this test program with your Bunny class definition.

```
public class BunnyTest
{
  public static void main(String[] args)
  {
    System.out.println("Testing Peter");
    Bunny peter = new Bunny();
    peter.displayInfo();
    peter.hop(12);
    peter.hop(12);
    peter.hop(9);
    peter.displayInfo();
    System.out.println("Testing Emily");
    Bunny emily = new Bunny();
    emily.displayInfo();
    emily.hop(9);
    emily.hop(9);
    emily.hop(9);
    emily.hop(12);
    emily.hop(9);
    emily.hop12();
    emily.displayInfo();
  }
}
```

```
> java BunnyTest
Testing Peter
This bunny is at position 10,10
This bunny has 5 carrot sticks remaining
hop
hop
hop
This bunny is at position 9,12
This bunny has 2 carrot sticks remaining
Testing Emily
This bunny is at position 10,10
This bunny has 5 carrot sticks remaining
hop
hop
hop
hop
hop
This bunny can't hop
This bunny is at position 6,11
This bunny has 0 carrot sticks remaining
>
```

More Bunnies

How could we keep track of a herd of bunnies?

We could make an array of bunnies.

More Bunnies

```
public class BunnyTest1
{
  public static void main (String[] args)
  {
    Bunny[] myBunnyHerd = new Bunny[10];

    myBunnyHerd[0] = new Bunny(3,6,4,"Foofoo");
    myBunnyHerd[1] = new Bunny(7,4,2,"Peter");
    myBunnyHerd[3] = new Bunny(9,2,3,"Ed");

    for(int i = 0; i < myBunnyHerd.length; i++)
    {
      if (myBunnyHerd[i] != null)
      {
        myBunnyHerd[i].hop(3);
        System.out.println(myBunnyHerd[i]);
      }
    }
  }
}
```

Even More Bunnies

Question 5: **[16 marks]**
The world desperately needs better bunny management software, so please help by writing a BunnyHerd class. A BunnyHerd object holds an array of Bunny objects. Your BunnyHerd class definition should include the following four methods:

constructor Expects two parameters, an integer representing the maximum number of bunnies in the herd, and a String for the name of the herd.

`addBunny(int xPos, int yPos, int carrots,String name)` Expects four parameters, the X- and Y-coordinates of the bunny, the number of carrots, and the name. This method creates a new Bunny object and stores the reference to the object in the next available location in the BunnyHerd object.

`deleteBunny(String name)` Expects one parameter, the name of the bunny. This method removes from the BunnyHerd object all references to bunnies with the given `name` by overwriting those references with the `null` pointer. This method does not change the pointer to the next available location in the BunnyHerd object.

`printHerd()` This method uses the `toString()` method of the Bunny object to print information about every Bunny in the herd.