University of Auckland

415.101

Java

Christian Collberg
September 12, 1997

Tutorial #7

This Week's Tutorial

- Using and defining elementary classes.
- Creating new class instances using **new**.
- Invoking methods.
- Referencing instance variables.
- Constructors.
- We will use a running example to illustrate these concepts: Computing and displaying scores in an athletic competition.

Classes as Object Templates

- A class serves as a *template* for objects.
- We can create new objects *of the class* using **new**. Eg., we can create a new **Integer** object using **new** (**Integer** is one of Java's built-in classes):

```
Integer I = new Integer(5);
```

- We can create our own classes and make new instances of them. Show the objects referenced by H and J below:

```
class Hello {int a = 5;}
Hello H = new Hello();
Hello J = new Hello();
```


Constructors

- A constructor is a method that is called when a new object is created.
- The name of a constructor is the same as the name of the class.
- Constructors usually initialize the object's instance variables.
- Show the objects referenced by H and J below:

```
class Hello {
   String a;
   Hello(String a) {this.a = a;}
}
Hello H = new Hello("5");
Hello J = new Hello("bye");
```

Invoking Methods

- Each class defines a number of methods. We can call these methods *through* an object.
- `this` refers to the current object.
- What output do the statements below produce?

```
class Counter {
   int c;
   Counter(int c)  {this.c = c;}
   void inc(int c) {this.c += c;}
   int value()     {return c;}
   void print()    {System.out.println(c);}
}
Counter a = new Counter(0);
Counter b = new Counter(5);
a.inc(5);
b.inc(a.value());
b.print();
```

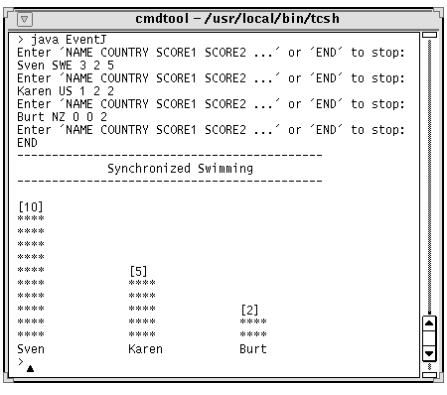

Private and Public

- Fields and methods can be declared `private` or `public`.
- `public` class members can be accessed from outside the class.
- `private` class members can only be accessed from within the class.
- All instance variables should be declared `private`.
- Methods will normally be declared `public`.

```
class Counter {
   private int c;
   Counter(int c) {this.c = c;}
   public void inc(int c) {this.c += c;}
   public int value() {return c;}
}
```


The `Event` Program

- Again, our task will be to write a program that reads in a number of competitors and their scores, totals the scores and prints out a histogram of the results.
- This time, though, we'll use classes and methods to break up the program in manageable chunks.



Which Classes Do We Need?

- Often, each class will represent the description of some "physical", real world, object.
- In our example, we're dealing with three such objects: competitions, competitors, and bar-graphs.
- Hence, our program will contain three classes in addition to the main class:

```
class Competition { ... }

class Contestant  { ... }

class BarGraph    { ... }

class EventJ {
  public static void main () { ... }
}
```

The Contestant Class I

What operations will we need to perform on contestants?

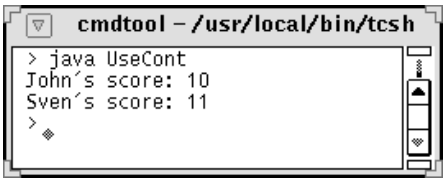
1. Create a new contestant, giving them a name and a country of origin. We do this using the `Contestant` *constructor*.
2. Add a new score to the contestant's total score.
3. Retrieve the contestant's total score, name, and country of origin.

```
class Contestant {
   Contestant (String name,
               String country) { ... }
   public String getName () { ... }
   public String getCountry () { ... }
   public int getTotal () { ... }
   public void addScore(int score){...}
}
```


The Contestant Class II

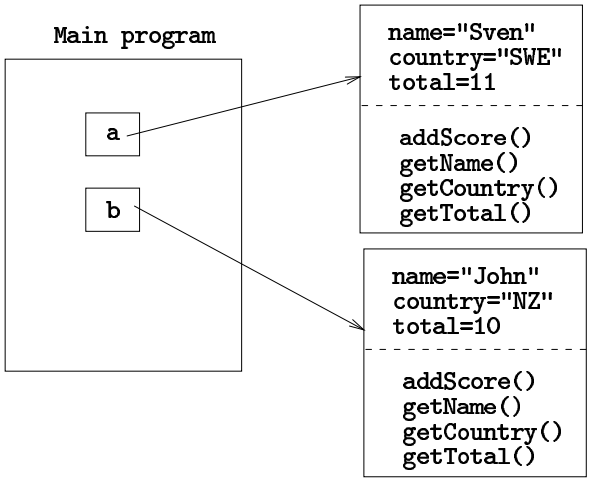
- Test the `Contestant` operations before using them in a real program:

```
class UseCont {
   public static void main (...) {
      Contestant a,b;
      a = new Contestant("John", "NZ");
      b = new Contestant("Sven", "SWE");
      a.addScore(1); a.addScore(9);
      b.addScore(3); b.addScore(8);
      System.out.println(a.getName() +
           "'s score: " + a.getTotal());
      System.out.println(b.getName() +
           "'s score: " + b.getTotal());
   }
}
```



The Contestant Class III

- What does the running program look like when we get to the end of the `main` method in the previous slide?



- Note that the running program contains two *objects*, one for each contestant. The variables `a` and `b` *refer to* (or *point to*) these objects.

The Contestant Class IV

```
class Contestant {
   private String name;
   private String country;
   private int    total;

   Contestant(String name,
              String country) {
      this.name       = name;
      this.country    = country;
      this.total      = 0;
   }
   public String getName () {
      return name; }
   public String getCountry () {
      return country; }
   public int getTotal () {
      return total; }
   public void addScore (int score) {
      total += score; }
}
```

The Competition Class I

What operations will we need to perform on competitions?

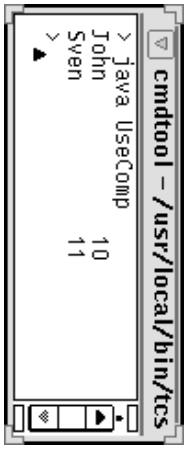
1. Create a new competition, giving it a name.
2. Add a new contestant to the competition.
3. Retrieve each contestant and the name of the competition.

```
class Competition {
   Competition (String name) { ... }
   public void addContestant (Contestant cont) { ... }
   public Contestant getContestant (int number) { ... }
   public int getNumberOfContestants () { ... }
   public String getName () { ... }
}
```


The Competition Class II

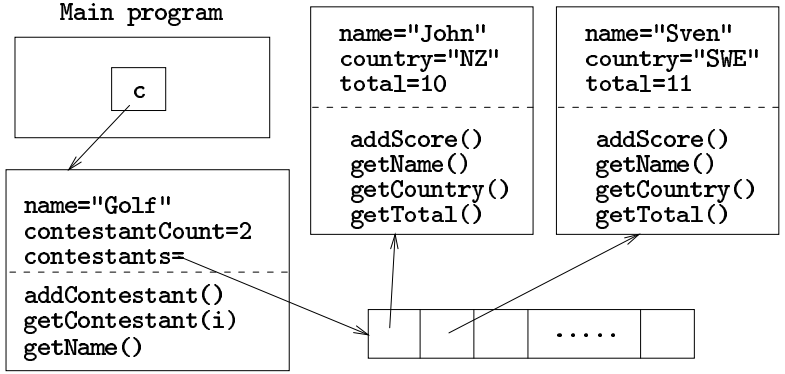
- Test the `Competition` class:

```
public static void main (String [] args) {
   Contestant a = new Contestant("John","NZ");
   a.addScore(1); a.addScore(9);
   Contestant b = new Contestant("Sven","SWE");
   b.addScore(3); b.addScore(8);

   Competition c = new Competition("Golf");
   c.addContestant(a); c.addContestant(b);

   for (int i=0;
        i<c.getNumberOfContestants(); i++)
      System.out.println(
         c.getContestant(i).getName() + "\t\t"
         + c.getContestant(i).getTotal());
}
```



The Competition Class III

- What objects exist when we reach the end of the `main` method in the previous slide?



The Competition Class IV (a)

```
class Competition {
   private static final int MaxNoOfContestants = 20;

   private Contestant contestants [];
   private int        contestantCount;
   private String     name;

   Competition (String name) {
      contestants = new Contestant [MaxNoOfContestants];
      contestantCount = 0;
      this.name = name;
   }
```

The Competition Class IV (b)

```
public void addContestant (Contestant cont) {
   contestants[contestantCount] = cont;
   contestantCount++; }

public Contestant getContestant (int number) {
   return contestants[number]; }

public int getNumberOfContestants () {
   return contestantCount; }

public String getName () { return name; }
}
```

Slide 7–16

The BarGraph Class I

What operations will we need to perform on bar graphs?

1. Create a new graph, giving it the "banner" (heading).
2. Add a new value to the graph, at the same time giving the label to print under the bar.
3. Print the graph.
4. We'll also use some private methods to simplify printing the graph.

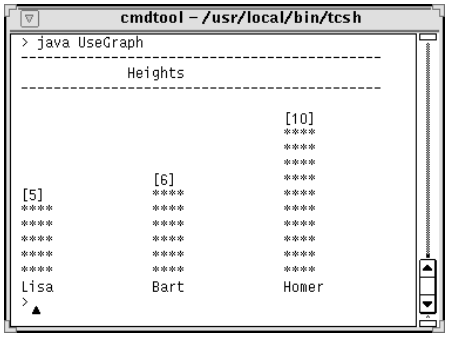
```
class BarGraph {
   BarGraph (String banner) {...}
   public void addValue (
      String label, int value) { ... }
   public void print () { ... }
}
```

Slide 7–17

The BarGraph Class II

- Test the `BarGraph` class:

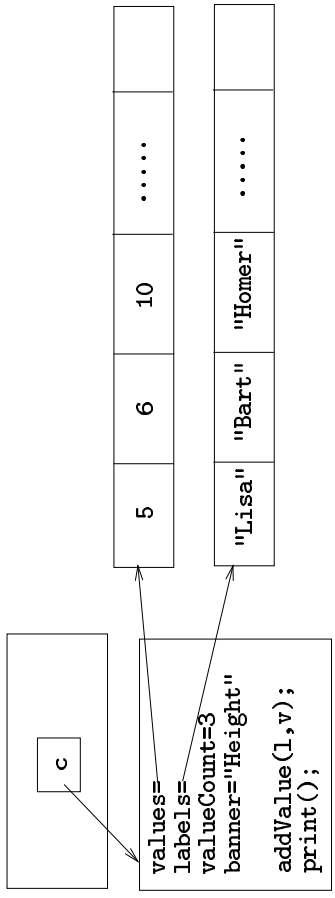
```
public static void main (String [] args) {
      BarGraph c = new BarGraph("Heights");

      c.addValue("Lisa", 5);
      c.addValue("Bart", 6);
      c.addValue("Homer", 10);

      c.print();
   }
```



Slide 7–18

The BarGraph Class III

- What objects does the program contain when we reach the end of the `main` method in the previous slide?



Slide 7–19

The BarGraph Class IV (a)

```
class BarGraph {
  private static final int MaxNoOfValues = 20;
  int    values [];
  String labels [];
  int    valueCount;
  String banner;

  BarGraph (String banner) {
    values     = new int [MaxNoOfValues];
    labels     = new String [MaxNoOfValues];
    valueCount  = 0;
    this.banner  = banner;
  }
```

Slide 7–20

The BarGraph Class IV (b)

```
public void addValue (
       String label, int value) {
   values[valueCount] = value;
   labels[valueCount] = label;
   valueCount++;
}

public void print () {
   printBanner();
   printBars();
   printLabels();
}

private int getMaxValue () {
  int maxValue=0;
  for (int c=0; c<valueCount; c++)
     maxValue = Math.max(maxValue,
                         values[c]);
  return maxValue;
}
```

Slide 7–21

The BarGraph Class IV (c)

```
private void printBanner () {
   System.out.println("-----------------------------------------");
   System.out.println("          " + banner);
   System.out.println("-----------------------------------------");
   System.out.println();
}

private void printLabels () {
  for (int c=0; c<valueCount; c++)
     System.out.print(labels[c] + "\t\t");
  System.out.println();
}
```

Slide 7–22

The BarGraph Class IV (d)

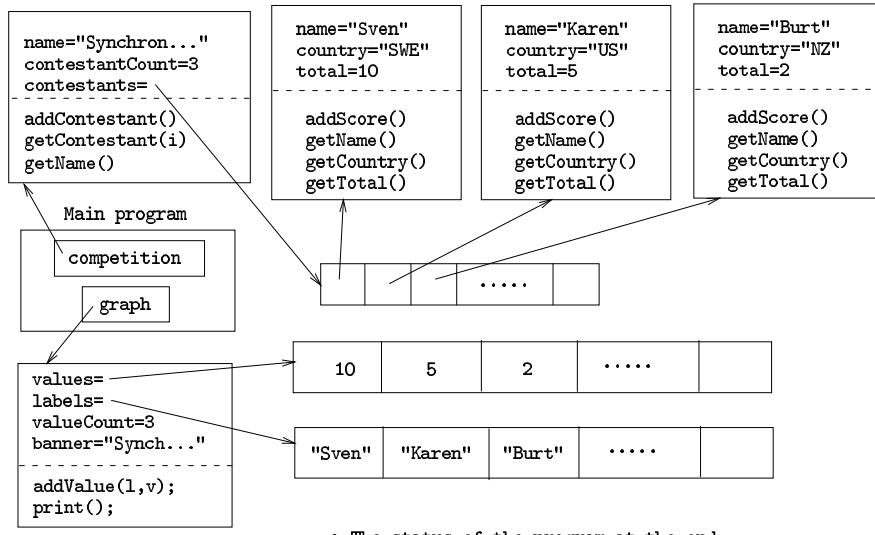
```
private void printBars () {
  for (int s=getMaxValue(); s>=0; s--){
     for (int c=0; c<valueCount; c++){
        if (values[c] == s)
           System.out.print("["+ values[c] +"]");
        else if (values[c] >= s)
           System.out.print("****");
        System.out.print("\t\t");
     }
     System.out.println();
  }
}
}
```

Slide 7–23

The Event Class I

1. First, read in a competition, ie. read in all the competitors name and scores.
2. Print the bar-graph.

```
class EventJ {
   private static Contestant readContestant() { ... }
   private static Competition readCompetition () { ... }
   private static void printBarGraph () { ... }
   public  static void main (String [] args) {
      Competition competition = readCompetition(
                                    "Synchronized Swimming");
      printBarGraph(competition);
   }
}
```

- The status of the program at the end of the main method.

The Event Class III (a)

```
private static Contestant readContestant() {
   String S = Input.toText("Enter 'NAME COUNTRY SCORE1 SCORE2 ...'" +
                           " or 'END' to stop:");
   StringTokenizer T = new StringTokenizer(S);
   String name = T.nextToken();
   if (name.equals("END")) return null;
   String country = T.nextToken();
   Contestant contestant = new Contestant(name, country);
   while (T.hasMoreTokens()) {
      int score = Integer.parseInt(T.nextToken());
      contestant.addScore(score);
   }
   return contestant;
}
```


The Event Class III (b)

```
private static Competition readCompetition (String Name) {
   Competition competition = new Competition(Name);
   for (;;) {
       Contestant contestant = readContestant();
       if (contestant == null) return competition;
       competition.addContestant(contestant); }
}
private static void printBarGraph (Competition competition) {
   BarGraph graph = new BarGraph(competition.getName());
   for (int c=0; c<competition.getNumberOfContestants(); c++)
      graph.addValue(competition.getContestant(c).getName(),
                     competition.getContestant(c).getTotal());
   graph.print();}
```

The Rational Class I

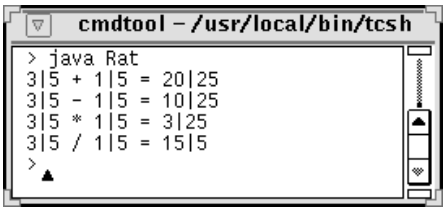
- Write a class `Rational` that allows us to perform arithmetic on rational numbers.
- A rational number is represented by two integers, the denominator and the numerator. We need operations for creating a new rational number, performing arithmetic on two rational numbers, and converting a rational number to a string (for printing).
- Show the state of the program after the `main` method has been called in the program on the next slide.

```
class Rational {
   Rational (int Denominator, int Numerator)
   public Rational add (Rational B)
   public Rational sub (Rational B)
   public Rational mul (Rational B)
   public Rational div (Rational B)
   public String toString ()
}
```

Slide 7–28

The Rational Class II

```
class Rat {
   private static void test (
      Rational a, String Op,
      Rational b, Rational R) {
      System.out.println(a+Op+b+" = "+R);
   }
   public static void main (...) {
      Rational a = new Rational(3,5);
      Rational b = new Rational(1,5);
      test(a, " + ", b, a.add(b));
      test(a, " - ", b, a.sub(b));
      test(a, " * ", b, a.mul(b));
      test(a, " / ", b, a.div(b));
}}
```



Slide 7–29

The Rational Class III

```
class Rational {
   private int D, N;
   Rational (int Denom, int Num) {D = Denom; N = Num; }
   public Rational add (Rational B) {
      return new Rational(D*B.N + N*B.D, N*B.N);}
   public Rational sub (Rational B) {
      return new Rational(D*B.N - N*B.D, N*B.N);}
   public Rational mul (Rational B) {
      return new Rational(D*B.D,N*B.N);}
   public Rational div (Rational B) {
      return new Rational(D*B.N,N*B.D);}
   public String toString () {
      return Integer.toString(D) + "|" + Integer.toString(N);}
}
```

Slide 7–30