CS 125 - Lecture 33

Objectives: recursion; graphics objects; selection sort; **Up next:** MP6 - due in 7 days;

Two types of recursion: forward vs. tail

1) **tail recursion** : The recursive call occurs at the very end of a method. The method executes **all** statements before jumping into the next recursive call.

2) **forward recursion** recursive call occurs before the end of the method.

1. **Compare: forward vs. tail**

```
public void tail(int n)
{
   if(n == 1)
      return;
   else
      System.out.println(n);

   tail(n-1);
}
```

```
public void forward(int n)
{
  if(n == 0)
     return;
  else
     forward(n-1);

   System.out.println(n);
```

2. **How to multiply all values together (spot the mistake):**

```
public int getMultiplierA() { // Use Forward Recursion
     if(next == null) return value;
     return value + next.getMultiplierA();
}
// Use TAIL accumulator recursion
public int getMultB(int result) {




}
```

3a. **Create an activation diagram on the right for** `prc(3, "*", true)`:

```
public static void prc(int c, String s, boolean newline) {
    if (newline && c==0) {
       System.out.println();
       return;
    }
    System.out.print(s);
    prc(c-1, s, newline);
}
```

3b. **How many stars are printed for:** `prc(3, "*", false)`?

4. **If each link has a larger value than the previous, will the following** `getMax()` **create a tree or chain of activations?**

```
class LinkedList{
   int value;
   LinkedList next;
}
public int getMax() {
   if (next == null)
      return value;  // BASE CASE

   int result = next.getMax();
   if (result < value) return value;
   else return next.getMax();
}
```

5. **Create an activation diagram for** `f3(31373)`**:**

```
public static int f3(int x) {
   if (x == 3) return 1;
   if (x < 10) return 0;

   return f3(x/10) + f3(x%10);
}
```

Desktop Apps:

6. **JFrame**

- Subclass of Container;
- Defines a rectangular area on screen to hold components (graphical objects like buttons, sliders, text labels, etc.)
- To use, import graphics packages:

```
import java.awt.*
import javax.swing.*
```

- Usage:

```
JFrame frame = new JFrame("Test Frame 1");
frame.setSize(200,100);
frame.setVisible( true );
frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
```

docs (hover):

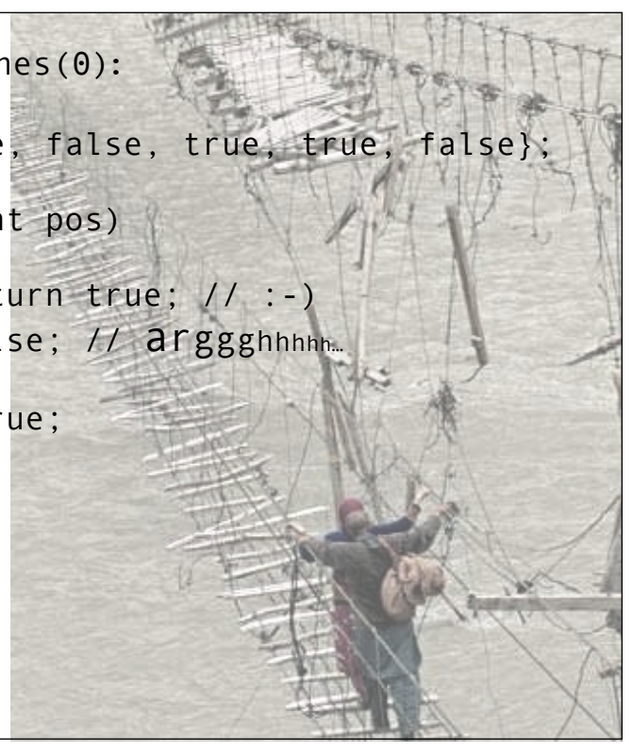
```
public void setBounds(int x, int y, int width, int height)
```

7. **Forward vs. tail recursion - examples:**

```
class LinkedList {
   private int value;
   private LinkedList next;
   public LinkedList(int v, LinkedList n) {
         value = v;
         next = n;
   }

// Use FORWARD Recursion
   int getMaxA() {
      if(next ==null) return value; // BASE CASE




   }

//Use TAIL Accumulator Recursion
   int getMaxB(int result) {




   }
```

8. **Create an activation diagram for** `jones(0)`:

```
static boolean[] steps = {true, false, true, true, false};

public static boolean jones(int pos)
// BASE CASES
  if( pos >= steps.length) return true; // :-)
  if( ! steps[pos]) return false; // arggghhhhh...
// RECURSIVE CASES
  if( jones(pos+2) ) return true;
  return jones(pos+1);
}
```



Hussaini bridge, Pakistan

9. **Selection Sort**

```
public class SelectionSort01 {
   // returns the INDEX of the minimum of array data
   // recursive function
   public static int findMin(double[] data, int lo, int hi) {









  }

   // swap two array values from posA to posB and vice-versa
   public static void swap(double[] data, int posA, int posB)
     {



     }

   // implement
   public static void sort(double[] data, int lo, int hi) {









   }
   public static void main(String[] args) {
      // sample data for testing ...
      double[] data = {21.,20.,18.,6.,15.,16.,17.,18.,19.};
      int pos =findMin(data,0,data.length-1);

      sort(data, 0, data.length-1);

      for (int i=0; i<data.length; i++) {
       System.out.println(data[i]);
      }
   }

}
```

CS 125 - Lecture 33

Objectives: intro to GUI; forward vs. tail recursion;

Up next: MP6 - due in 1 week; Midterm 3 next Wednesday;

Discuss: Have you used a phone book?

1. **Two types of recursion: forward vs. tail**

1) **tail recursion** : The recursive call occurs at the very end of a method. The method executes **all** statements before jumping into the next recursive call.

2) **forward recursion** recursive call occurs before the end of the method.

2. **Compare: forward vs. tail**

```
factorial(int n) {
  if (n == 0) return 1;
  return n * factorial(n - 1);
}
```

```
factorial1(int n, int accumulator) {
  if (n == 0) return accumulator;
  return factorial1(n - 1, n * accumulator);
}
factorial(n) {
  return factorial1(n, 1);
}
```

3. **Forward vs. tail recursion - examples:**

```
class LinkedList {
   private int value;
   private LinkedList next;
   public LinkedList(int v, LinkedList n) {
         value = v;
         next = n;
   }

// Use FORWARD Recursion
   int getMaxA() {
      if(next ==null) return value; // BASE CASE
```

```
//Use TAIL Accumulator Recursion
   int getMaxB(int result) {
```

```
   }
}
```

4. **How to multiply all values together (spot the mistake):**

```
public int getMultiplierA() { // Use Forward Recursion
     if(next == null) return value;
     return value + next.getMultiplierA();
}

// Use TAIL accumulator recursion
public int getMultB(int result) {









}
```

6. **JFrame**

- Subclass of Container;
- Defines a rectangular area on screen to hold components (graphical objects like buttons, sliders, text labels, etc.)
- To use, import graphics packages:

```
import java.awt.*
import javax.swing.*
```

- Usage:

```
JFrame frame = new JFrame("Test Frame 1");
frame.setSize(200,100);
frame.setVisible( true );
frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
```

docs (hover):

```
public void setBounds(int x, int y, int width, int height)
```


- Subclass of Container;
- Defines a rectangular area on screen to hold components (graphical objects like buttons, sliders, text labels, etc.)
- To use, import graphics packages:

```
import java.awt.*
import javax.swing.*
```

- Usage:

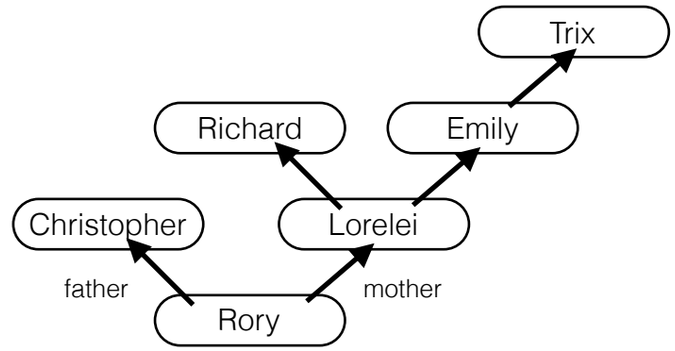
```
JFrame frame = new JFrame("Test Frame 1");
frame.setSize(200,100);
frame.setVisible( true );
frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
```

docs (hover):

```
public void setBounds(int x, int y, int width, int height)
```

6. **Person class (for a family tree?):**

```
class Person {
   private String name;
   private Person mother;
   private Person father;

//  Write setters and getters (read/write) for each instance variable
//  Write convenience methods that get the
//  mother's name and the father's name for the person

//  A constructor that takes a String : newName


}
```



7. Set up the people and the relationships shown above:

```
public static void main(String[] args) {




}
```

8. Assume the head of the family tree is female. Write a *forward* recursive method *getFL1* that returns a string of the entire female lineage of person '*p*'. Insert commas between each person's name and a period at the end.

"*name,mother,grand-mother,great-grand-mother,….*"

```
   public String getFL1() {




 }
```

9. **Create an activation diagram for** `f3(31373)`**:**

```
public static int f3(int x) {
   if (x == 3) return 1;
   if (x < 10) return 0;

   return f3(x/10) + f3(x%10);
}
```