CSE 8B Today

• Exam 4 next Tuesday 27th
Review on DISCUSSION tomorrow! (and review sheet)

• PSA EXTRA (optional): *AI player*
  - Hints today AFTER the break to give exam3 back.
  - Surprise! It’s about RECURSION 😊
  - It’ll replace your lowest **PSA2 to PSA7** score. BUT YOU SHOULD DO IT FOR THE SAKE OF LEARNING! and practice with recursion
  - It’s very challenging BUT fun!

X to move.
Is there a way to win?

MORE RECURSION!

FINAL EXAM:
- Mandeville Auditorium
- Post request NOW if you need to *change to the other section time*
Recursion, more *meaningful* practice: FIND!

Write a method to find an element in an array *between startIndex and the end* of the array and return its index as shown in the examples:

```java
public static int find( String[] myList, String toFind, int startIndex )
{
    ???
}
```

```java
>> int[] a = {"Hello", "Welcome", "Turtle", "fun"};
>> RecursionDemo.find( a, "Hello", 0 )
0
>> RecursionDemo.find( a, "Happy", 0 )
-1
>> RecursionDemo.find( a, "fun", 2 )
3
>> RecursionDemo.find( a, "Hello", 2 )
-1
```

What is the base case?
A. The element at startIndex is equal to toFind
B. The array is empty
C. startIndex is (greater than or) equal to myList.length
D. toFind is not in myList
E. More than one of these
Exam 3 - results
public abstract class Cake {
    protected boolean frosting; // field
    public abstract String recipe(); // abstract method
    public String toString()
    {
        return "Something sweet!";
    }
} // END CLASS CAKE

public class ChocolateCake extends Cake {
    public ChocolateCake (boolean addFrosting)
    {
        this.frosting = addFrosting;
    }
    public String recipe()
    {
        return "Prepare the dough and bake.";
    }
    public String toString()
    {
        if (this.frosting){ return super.toString()+ " with frosting and chocolate";}
        else{ return super.toString()+ " with chocolate"; }
    }
} // END CLASS CHOCOLATECAKE

public class BirthDayChocoCake extends ChocolateCake {
    protected int nCandles; // additional field
    public BirthDayChocoCake (boolean addF, int numCandles)
    {
        super(addF); this.nCandles = numCandles;
    }
    public void putCandles()
    {
        System.out.println("The cake has "+ nCandles +" candles");
    }
} // END
public abstract class Cake {
    protected boolean frosting; // field
    public abstract String recipe(); // abstract method
    public String toString()
    {
        return "Something sweet!"; } } // END CLASS CAKE

public class ChocolateCake extends Cake {
    public ChocolateCake (boolean addFrosting)
    { this.frosting = addFrosting; }
    public String recipe()
    { return "Prepare the dough and bake."; }
    public String toString()
    {
        if (this.frosting)
            return super.toString() + " with frosting and chocolate";
        else
            return super.toString() + " with chocolate";
    }
} // END CLASS CHOCOLATECAKE

public class BirthdayChocoCake extends ChocolateCake {
    protected int nCandles; // additional field
    public BirthdayChocoCake (boolean addF, int numCandles)
    { super(addF); this.nCandles = numCandles; }
    public void putCandles()
    { System.out.println("The cake has " + nCandles + " candles"); }
} // END

<table>
<thead>
<tr>
<th></th>
<th>Compilation Error (A)</th>
<th>Runtime Error (B)</th>
<th>No error (C)</th>
</tr>
</thead>
</table>
| ChocolateCake c3 = new ChocolateCake( true );
((BirthdayChocoCake) c3).putCandles(); |                      |                  |             |
| Cake c4 = new BirthdayChocoCake( false, 5 );
String message = c4.recipe(); |                      |                  |             |
| BirthdayChocoCake c = new ChocolateCake(true); |                      |                  |             |
| Cake c2 = new Cake( false ); |                      |                  |             |
public abstract class Cake {
    protected boolean frosting;  // field
    public abstract String recipe();  // abstract method
    public String toString()
    {    return "Something sweet!";    }    }    } // END CLASS CAKE

public class ChocolateCake extends Cake {
    public ChocolateCake (boolean addFrosting)
    { this.frosting = addFrosting; }
    public String recipe()
    {    return "Prepare the dough and bake.";    }
    public String toString()
    {    if (this.frosting){ return super.toString()+ " with frosting and chocolate";}
          else{ return super.toString()+ " with chocolate";  }
    }  }  } // END CLASS CHOCOLATECAKE

public class BirthDayChocoCake extends ChocolateCake {
    protected int nCandles;  // additional field
    public BirthDayChocoCake ( boolean addF, int numCandles)
    { super(addF); this.nCandles = numCandles; }
    public void putCandles()
    { System.out.println("The cake has "+ nCandles +" candles"); }  }  } // END

class MyTester {
    public static void main( String[] args ) {
        Cake c1 = new ChocolateCake( true );
        Cake c2 = new BirthDayChocoCake( false, 3 );
        ( (BirthDayChocoCake) c2).putCandles();
        System.out.println( c2 );
        System.out.println( c1 );    }    }
public abstract class Cake {
    protected boolean frosting; // field
    public abstract String recipe(); // abstract method
    public String toString()
    { return "Something sweet!"; } } // END CLASS CAKE

public class ChocolateCake extends Cake {
    public ChocolateCake (boolean addFrosting)
    { this.frosting = addFrosting; }
    public String recipe()
    { return "Prepare the dough and bake."; }
    public String toString()
    { if (this.frosting){ return super.toString()+ " with frosting and chocolate";} else{ return super.toString()+ " with chocolate"; }
    } } // END CLASS CHOCOLATECAKE

public class BirthDayChocoCake extends ChocolateCake {
    protected int nCandles; // additional field
    public BirthDayChocoCake (boolean addF, int numCandles)
    { super(addF); this.nCandles = numCandles; }
    public void putCandles()
    { System.out.println("The cake has "+ nCandles +" candles"); } } // END

class MyTester {
    public static void main( String[] args ) {
        Cake c1 = new ChocolateCake( true );
        Cake c2 = new BirthDayChocoCake( false, 3 );
        (BirthDayChocoCake) c2).putCandles();
        System.out.println( c2 );
        System.out.println( c1 );
    } }

Which toString method I’ll call when I print c2?
(REMEMBER print command AUTOMATICALLY calls toString)

A. None, BirthDayChocoCake does NOT have that method
B. toString method from ChocolateCake
C. toString method from Cake
import javax.swing.*;
import java.awt.*;

public class MyPanel extends JPanel {
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawLine(10, 10, 100, 100);
        g.drawLine(100, 100, 190, 10);
    }

    public static void main(String[] args) {
        JFrame f = new JFrame();
        f.setSize(200, 200);
        JPanel p = new MyPanel();
        f.add(p);
        f.setVisible(true);
    }
}

What does super.* mean? (* being any method or variable name that exists)
A. Access something in the calling object
B. Access something in the parent class
C. Override something in the parent class

True False This code will display the second window (the empty one), because only the JFrame is visible.
True False At runtime, Java will detect that the object referenced by p is actually a MyPanel object and will use the paintComponent method in the MyPanel class.
True False If we remove the line f.add( p ); the JPanel p won’t be added to of our JFrame f and therefore we won’t see the V drawing in the window displayed after running the main method.
True False The line super.paintComponent(g) included in MyPanel’s paintComponent method tells Java that this method overrides the paintComponent method from JPanel. i.e., without that line, in this code Java would use the paintComponent method from the JPanel class instead of the one from the MyPanel class.
In the sample code from previous question, if we want to assign something to the variable \( p \), we can assign a reference to an object of the same class than \( p \), or to an object of any \textbf{SUBCLASS}\(^*\) of JPanel and it will be correct; this is know as \textbf{POLYMORPHISM} (or any close “spelling” ;-) \textbf{INHERITANCE} enables you to define a general class (e.g., a superclass) and later extend it to more specialized classes (e.g., subclasses). – \textit{directly from the book...}

In the code from previous question, MyPanel is a \textbf{SUBCLASS}\(^*\) of JPanel, and JPanel is the \textbf{SUPERCLASS}\(^*\) of MyPanel. This implies that the class \texttt{MYPANEL} inherits fields and methods from the class JPanel.

Every class in Java has a common superclass, the \texttt{OBJECT} class. \texttt{(Java.lang.Object is also ok!)}

\* \textit{OR ANY SYNONIM from the BOOK}
An object is structured data that is alive, responsible, and intelligent.

This week’s objects and classes will be clever than ever . . .

This part and assignment are OPTIONAL!

- DUE ON TUESDAY (no additional slip day, everyone due on TUESDAY)
- INDIVIDUAL
- REPLACE your lowest PSA2 – PSA7 score

Is there a way to win?

Slides from Zach Dodds at Harvey Mudd College