CSE 8B Today

Don’t forget: Fill out your CAPES, TA evaluations, and your survey

Just 2 more days of class!

Today: More Recursion and Exceptions
• Generics – quick example
• Thread example
• Exceptions
• Review midterm 4 – recursion examples

Thursday: Wrap up and review

How’s PSA8 going?
A. I haven’t started at all
B. I’ve read it and started thinking about it
C. I’ve started some coding
D. I’m done
1. What is a try block used for?

A. It runs whenever there is a compile error.

B. It runs when exceptions are thrown at runtime.

C. It tries to run code that might throw an exception.

D. It tries to suppress any compile errors within the block.
2. Which class to extend when you want to define your own exceptions?

A. Exception
B. CustomException
C. Throws
D. TryCatch
3. Generic classes are defined with a formal generic type, e.g. E, `java.util.ArrayList<E>` which can be replaced later with an actual concrete type. Replacing a generic type is called:

A. Subtype definition

B. Runtime type

C. A generic instantiation

D. Object casting
4. Is this definition of an ArrayList correct? Why?

ArrayList<char> charList = new ArrayList<char>();

A. Yes, it’s correct because ArrayList is a generic class that can be declared using any type.

B. Yes, it’s correct because we are putting the same type in both sides of the assignment.

C. No, it’s NOT correct, because we can’t replace the generic type from ArrayList<T> with a primitive type.

D. No, it’s NOT correct, because if we want to use a set of chars, we should always use a String.
CSE 8B Today

FINAL EXAM – TUESDAY and THURSDAY in MANDEVILLE AUDITORIUM
• Seating assignment on your email on MONDAY – come early to find it (specially if you don’t get your email)
• The final exam will look a lot like the in-term exams, only longer. Designed to take about 2 hours (you will have 3)
Generics: example with ArrayList<T>

Binary Search example

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>14</th>
<th>16</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

The method will return the *position* of the item if it is found

```java
public static int binarySearch( ArrayList<Integer> theList,
                               int toFind, int low, int high)
```

low will start at 0, high will start at theList.length-1

Assuming the array pictured above is named `myA`...

```java
> binarySearch( myA, 4, 0, 8 )

> binarySearch( myA, 5, 0, 8 )
```
Generics

1. In the statement “ArrayList<T>”, what does T represent?
   A. A type
   B. A parameter
   C. That T thingy
   D. An Object

   Here, <T> represents a formal generic type, which can be replaced later with an actual concrete type.
   Replacing a generic type is called a generic instantiation.

2. Do the following compile? Why or Why Not?
   A. ArrayList<T> al= new ArrayList<Integer>();
   B. ArrayList<T> al= new ArrayList<T>();
   C. ArrayList<Integer> al = new ArrayList<Integer>(); → THIS IS THE ONLY THAT IS OK
   D. ArrayList<int> al = new ArrayList<int>();

   Generic types must replaced with reference types.
   You cannot replace a generic type with a primitive.

3. What do you think that is roughly the same as ArrayList al= new ArrayList(); ?
   A. ArrayList<T> al= new ArrayList<T>();
   B. ArrayList<Object> al= new ArrayList<Object>();
   C. ArrayList<Integer> al= new ArrayList<Integer>();
Threads ...
PanelAnimator is an inner class of ConnectFourAnimationDemo class
PanelAnimator implements Runnable {
    int panelNum;
    int delay;
    public PanelAnimator( int num, delayIn ) {
        panelNum = num;
        delay = delayIn;
    }
    public void run() {
        try {
            while( true ) {
                if ( running ) {
                    if ( board[panelNum] == ' ' )
                        board[panelNum] = turn;
                    else
                        board[panelNum] = ' ';
                }
                repaint();
                Thread.sleep( delay );
            }
        } catch (InterruptedException ex) { } 
    }
}

What will the GUI do when the user clicks “Start”?
A. All three circles will change colors automatically, like they did before.
B. Only the first circle will change colors automatically.
C. Nothing will happen.
class PanelAnimator implements Runnable {
    int panelNum;
    int delay;
    public PanelAnimator( int num, delayIn ) {
        panelNum = num;
        delay = delayIn;
    }
    public void run() {
        try {
            while( true ) {
                if ( running ) {
                    if ( board[panelNum] == ' ' )
                        board[panelNum] = turn;
                    else
                        board[panelNum] = ' ';
                    repaint();
                    Thread.sleep( delay );
                }
                Thread.sleep( delay );
            }
        } catch (InterruptedException ex) { }
    }
}

// In the ConnectFourAnimationDemoConstructor...
PanelAnimator p0 = new PanelAnimator( 0, 500 );
PanelAnimator p1 = new PanelAnimator( 1, 100 );
PanelAnimator p2 = new PanelAnimator( 2, 300 );

new Thread(p0).start();
new Thread(p1).start();
new Thread(p2).start();

DO IT AT HOME: Add code to make the three circles change color at different speeds

How many threads do we have running with this program?
A. 0
B. 1
C. 2
D. 3
E. 4
What is an IOException??

Essentially, an exception is just an error that can be passed around (thrown).
If sourceFile does not exist, the Scanner’s constructor will throw an exception.
// In class StringPlay
public String readIt( String filename )
{
    File sourceFile = new File( filename );
    Scanner input = null;
    try {
        input = new Scanner( sourceFile );
    }
    catch ( IOException e ) {
        System.out.println( e.getMessage() );
        return "";
    }
    String allText = "";
    while ( input.hasNextLine() )
    {
        String s1 = input.nextLine();
        allText = allText.concat( s1 );
    }
    System.out.println( allText );
    return allText;
}

1. IF something happens here …
2. And it’s actually this kind of exception…
3. DO THIS! (e.g., get out!)
Handling exceptions

// In class StringPlay
public String readIt(String filename)
{
    File sourceFile = new File(filename);
    Scanner input = null;
    try {
        input = new Scanner(sourceFile);
    }
    catch (IOException e) {
        System.out.println(e.getMessage());
        // return ""; NO MORE RETURN HERE
    }
    String allText = "";
    while (input.hasNextLine())
    {
        String s1 = input.nextLine();
        allText = allText.concat(s1);
    }
    System.out.println(allText);
    return allText;
}

What happens if we remove the return from within the catch block (and call the method with a bad filename)?
A. The method will throw a NullPointerException
B. The method will return null
C. Nothing different will happen (it will still return the empty string and print the error message)
Handling exceptions

// In class StringPlay
public String readIt( String filename )
{
    File sourceFile = new File( filename );
    Scanner input = null;
    try {
        input = new Scanner( sourceFile );
    }
    catch ( IOException e ) {
        System.out.println( e.getMessage() );
        // return ""; NO MORE RETURN HERE
    }
    String allText = "";
    while ( input.hasNextLine() )
    {
        String s1 = input.nextLine();
        allText = allText.concat( s1 );
    }
    System.out.println( allText );
    return allText;
}

Why will this code compile, since it appears to throw an exception which is not caught or thrown?

A. It will only compile if we give the java compiler a special flag
B. A NullPointerException is an unchecked exception which doesn’t need to be caught
C. The exception is thrown within the try/catch block that is already in the code
Handling exceptions

// In class StringPlay
public String readIt( String filename )
{
    File sourceFile = new File( filename );
    Scanner input = null;
    try {
        input = new Scanner( sourceFile );
    }
    catch ( IOException e ) {
        System.out.println( e.getMessage() );
        // return ""; NO MORE RETURN HERE
    }
    String allText = "";
    while ( input.hasNextLine() ) {
        String s1 = input.nextLine();
        allText = allText.concat( s1 );
    }
    System.out.println( allText );
    return allText;
}

Why will this code compile, since it appears to throw an exception which is not caught or thrown?

A. It will only compile if we give the java compiler a special flag
B. A NullPointerException is an unchecked exception which doesn’t need to be caught
C. The exception is thrown within the try/catch block that is already in the code

SOME EXCEPTIONS ARE “UNCHECKED” (i.e., they don’t need to be caught)
Practicing PROBLEM: Complete the code to prompt the user to enter a new filename until the file exists.

```java
public String readIt( String filename ) {
    File sourceFile = new File( filename );
    Scanner input = null;
    Scanner userInput = new Scanner( System.in );
    while ( input == null ) {
        try {
            input = new Scanner( sourceFile );
        } catch ( IOException e ) {  
            System.out.println( e.getMessage() );
        }
        catch ( IOException e ) {  
            System.out.println( e.getMessage() );
        }
    }
    String allText = "";
    while ( input.hasNextLine() )
    {
        String s1 = input.nextLine();
        allText = allText.concat( s1 );
    }
    System.out.println( allText );
    return allText;
}
```
So EXCEPTIONS are ...

RuntimeException, Error, and their subclasses are known as *unchecked exceptions*.

All other exceptions are known as *checked exceptions* (i.e. the compiler forces you to check and deal with them in a try-catch block or throw them in the method header)

OBJECTS! (surprise 😊) of one of these classes (or subclasses)
Java Exceptions: Main Goal

• When you encounter a situation that won’t let your code act “normally”
  – Don’t “print an error” (printing to the screen isn’t the only way we interact with computers)
  – Throw an exception back to the location that called this method
  – It will then catch it and decide what to do

• Why throw it?
  – Scanner (written by Sun) doesn’t get to dictate the way YOU handle an exceptional circumstance
public class TurtleTester {
    public static void mosaic(Turtle t, int side){
        if (side >0 && side <= 300) {
            t.square(side);
            t.forward(20);
            mosaic(t, side*2);
            System.out.println("Square side: " + side);
        }
        return;
    } // end of mosaic method

    public static void main( String[] args ) {
        World w = new World( 400, 400 );
        //Create a turtle in w at (x,y)
        Turtle jose = new Turtle( 200, 400, w );
        TurtleTester.mosaic( jose, 40);  
    } // end of main method
}

How many stack frames do we create when we run this main?
A. 1
B. 3
C. 5
D. 7
public class TurtleTester {
    public static void mosaic(Turtle t, int side) {
        if (side > 0 && side <= 300) {
            t.square(side);
            t.forward(20);
            mosaic(t, side * 2);
            System.out.println("Square side: "+side);
        }
        return;
    } // end of mosaic method

    public static void main(String[] args) {
        World w = new World(400, 400);
        // Create a turtle in w at (x,y)
        Turtle jose = new Turtle(200, 400, w);
        TurtleTester.mosaic(jose, 40); // end of main method
    } // end of CLASS TurtleTester
public class TurtleTester {
    public static void mosaic(Turtle t, int side) {
        if (side > 0 && side <= 300) {
            t.square(side);
            t.forward(20);
            mosaic(t, side * 2);
            System.out.println("Square side: " + side);
        }
        return;
    } // end of mosaic method

    public static void main(String[] args) {
        World w = new World(400, 400);
        // Create a turtle in w at (x,y)
        Turtle jose = new Turtle(200, 400, w);
        TurtleTester.mosaic(jose, 40);
    } // end of main method

What will be the order of the numbers printed with squared side: ... ?
A. 40, 80, 160, 320
B. 320, 160, 80, 40
C. 40, 80, 160,
D. 160, 80, 40
public class TurtleTester {
    public static void mosaic(Turtle t, int side) {
        if (side > 0 && side <= 300) {
            t.square(side);
            t.forward(20);
            mosaic(t, side * 2);
        }
        System.out.println("Square side: "+side);
    }
    // end of mosaic method
    public static void main(String[] args) {
        World w = new World(400, 400);
        // Create a turtle in w at (x,y)
        Turtle jose = new Turtle(200, 400, w);
        TurtleTester.mosaic(jose, 40);
    } // end of main method
} // end of CLASS TurtleTester

BASE CASE: or "how to stop calling myself!"
side <= 0 || side > 300

Square side: 160
Square side: 80
Square side: 40
**Exam 4 – P2**

**ASSUME** i is NEVER a negative number

```java
public int addAllElements( int[] numArray, int i) {
    if (I_DONT_WANT_TO_ADD_THIS_ELEMENT) { // out of bounds?
        return 0;
    }
    else { // recursive case
        RETURN!!! first number + SUM OF THE REST
    }
} // END addAllElements
```

What is OUT OF BOUNDS here?

A. i < numArray
B. i >= numArray.length
C. I == 0
D. numArray.length == 0

What is the “sum of the rest”?

A. addAllElements (numArray, i-1)
B. addAllElements (numArray, i+1)
C. addAllElements (numArray, i)
D. addAllElements (numArray, 0)
ASSUME i is NEVER a negative number

```
public int addAllElements(int[] numArray, int i) {
    if ( i >= numArray.length) {
        return 0;
    }
    else { // recursive case
        return numArray[i] + addAllElements(numArray, i+1);
    }
} // END addAllElements
```
Exam 4 – P3

```java
public ButtonsFrame() {
    // constructor
    // code to create and add buttons to JFrame
    // WRITE HERE the code to create a MyButtonListener and add it
    // to each of the two JButton defined (jbt1 and jbt2)
    MyButtonListener bl = new MyButtonListener(); //1)
    jbt1.addMouseListener(bl); //2)
    jbt2.addMouseListener(bl); //3)
    
    // END OF ClickingFrame
}
```

Can I also do any of this options?

A. MouseListener bl = new MyButtonListener(); in 1)
B. JButton.addActionListener( new MyButtonListener() ); in 2)
C. JButton.addMouseListener( new MyListener() ); in 3)
D. All of the above
E. None of the above

/* WRITE THE CLASS MyButtonListener, which should correctly implement the interface
MOUSELISTENER, and print (in the terminal, using the standard println method) a different
message for each of its Mouse events. */
class MyButtonListener implements MouseListener {
    ...
}
... public ButtonsFrame() {  // constructor
  ... // code to create and add buttons to JFrame
  // WRITE HERE the code to create a MyButtonListener and add it
  // to each of the two JButton defined (jbt1 and jbt2)
  MyButtonListener bl = new MyButtonListener();
  jbt1.addMouseListener(bl);
  jbt2.addMouseListener(bl);
  ...
}
}  // END OF ClickingFrame

/* WRITE THE CLASS MyButtonListener, which should correctly implement the interface MOUSELISTENER, and print (in the terminal, using the standard println method) a different message for each of its Mouse events. */
class ButtonListener implements MouseListener {
  public void mouseClicked( MouseEvent e ) { System.out.println( "Clicked on a button" ); }
  public void mousePressed( MouseEvent e ) { System.out.println( "Mouse pressed" ); }
  public void mouseReleased( MouseEvent e ) { System.out.println( "Mouse released" ); }
  public void mouseEntered( MouseEvent e ) { System.out.println( "Mouse entered" ); }
  public void mouseExited( MouseEvent e ) { System.out.println( "Mouse exited" ); }
}
CSE 8B Today

FINAL EXAM – TUESDAY and THURSDAY in MANDEVILLE AUDITORIUM

• Seating assignment on your email on MONDAY – come early to find it (specially if you don’t get your email)
• The final exam will look a lot like the in-term exams, only longer. Designed to take about 2 hours (you will have 3)

The best things you can do to review or get help before the final:

• Study by RE-DOING clicker questions, in-class coding and PSAs (in paper??)!!

• Check website supplementary material, videos, ...

• Office hours!
  • LAST DAY I’ll be doing office hours: Monday morning.
  • Tutors and TA’s will also be available until Monday (double check calendar) After Monday, only Piazza.

• REVIEW SESSION: at discussion this week AND Thursday lecture.