CS 367 Announcements
Tuesday, February 2, 2015

Homework h1 due 10 pm Friday, February 6th
Program p1 due 10 pm Sunday, February 15th (get started now!)

We'll discuss assignment submission Thursday.
Assignment questions? Post on Piazza or see a TA during lab consulting hours.

Last Time

Iterators

- coding iterator interfaces and the Java API
- using iterators
- making a class iterable
- options for implementing iterators

Today

Exceptions Review

- throwing
- handling
- execution
- throws and checked vs. unchecked
- defining
- practice

Next Time

Read: finish *Exceptions*, start *Linked Lists*
Primitive vs. Reference Types

- assignment
- parameter passing

ADTs vs. Data Structures
Chains of Linked Nodes
Submitting Work

Exception Throwing – Signaling a Problem

When a problem is detected, we (or the runtime environment) can signal that probelm by throwing an expcetion

!!!:
When an exception is thrown, normal execution ends, we switch to exception handling mode!

Java Syntax

```
throw exceptionObject;
```

key word

Or we can construct a new exception object

Example

```
/ / You can add this to ArrayBag's remove( )

/ / no items can be remove if numItems is smaller than 0
if ( numItems < = 0 )  / / detect step
        throw new EmptyBagException ( );  / / signal the problem

/ / we will learn how to make our own exception classes
```

Exception Handling – Resolving a Problem

with "try-catch" statement

1. "try" indicate the code might be problematic
2. "catch" exceptions to handle particular problem

Java Syntax

```
try {
   // try block
   code that might cause an exception to be thrown

} catch (ExceptionType1 identifier1) {
   // catch block
   code to handle exception type 1

} catch (ExceptionType2 identifier2) {
   // catch block
   code to handle exception type 2

}
... more catch blocks

finally {
   // finally block - optional
   code always executed when try block is entered
}
```

→ exception handler

"catches" : 0 or more

Don't put "return" or "throw" statement here!

Example

```
/ / In the main method in the ArrayBag tester

try {
      while ( true )
            bag1.remove ( );
} catch ( EmptyBagException e ){
      System.out.println( "Remove method resulted in an EmptyBagException" );
}
```

Exception Execution

Normal Execution

- Start in normal executing mode on the top of the main ()
- Execute all normal code, including code in the “try" and “finally” blocks
- Skip code in catch blocks
- Switch to exception handling mode, and when an exception is thrown
    - then it is entirely different mode…

activation record = stack frame

Exception Execution

- In this mode, we are going to skip all the “normal” code
    - including code in the remaining “try” blocks that is entered

- We are still going to execute the code in the “finally” block though
    - but if the corresponding “try" block is entered

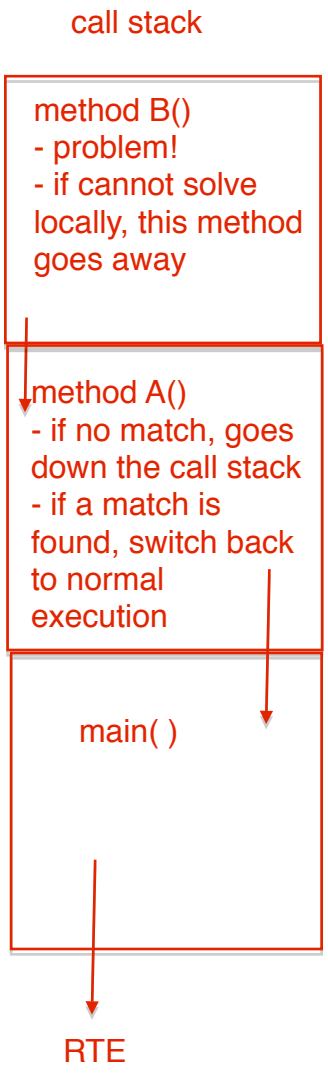
-> Search for matching catch in this order:

1. locally: if the exception was thrown by code in the “try” block
   look at its “catch” block for match

2. remotely: look down the call stack for a match
   can only use matching catch block
   if the problematic method call is in the corresponding try block

    - a) if a match is found
        - switch back to normal execution
        - execute code in catch block
        - execute code in the corresponding finally block
        - continue executing after “try-catch” block

    - b) if match not found
        - execute code in finally blocks for any try block entered
        - In the end, main () throws the execution to the RTE
            - which terminates your program
            - and display an exception message



`throws` clause – Passing the Buck

Checked vs. Unchecked

UNCHECKED
- For problems that can and should be avoided with careful programming
- e.g. NullPointerException, IndexOutofBoundException, etc

CHECKED
- For unavoidable problems
- e.g. IOException, FileNotFoundException

- Complier checks that the coder is aware of the problem by looking for…
  - a catch block
  - or a throws clause

Java Syntax

```
... methodName(parameter list)
       throws ExceptionType1, ExceptionType2, ... {
    ...
}
```

comma separated list for possible exceptions

Example

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

Defining a New Exception Class

Checked

```
public class MyException extends Exception {

      / / CHECKED
}
```

Unchecked

```
public class MyException extends RunTimeException {
        / / UNCHECKED

}
```

Example example for creating checked exception class that can store a msg

```
public class EmptyBagException extends Exception {

    public EmptyBagException() {
        super();   / / call the super class constructor
    }

    public EmptyBagException(String msg) {
        super(msg);
    }

}
```

ExceptionTester Example

```
public class ExceptionTester {

   public static void main(String[] args) {
      System.out.print("main[");
      try {
         methodA( ); System.out.print("after A,");
         methodE( ); System.out.print("after E,");
      } catch (RedException exc) {
         System.out.print("main-red,");
      } catch (GreenException exc) {
         System.out.print("main-green,");
      } finally {
         System.out.print("main-finally,");
      }
      System.out.println("]main");
   }

   private static void methodA( ) {
      System.out.print("\nA[");
      try {
         methodB( );
         System.out.print("after B,");
      } catch (BlueException exc) {
         System.out.print("A-blue,");
      }
      System.out.println("]A");
   }

   private static void methodB( ) {
      System.out.print("\nB[");
      methodC( );
      System.out.print("after C,");
      try {
         methodD( );
         System.out.print("after D,");
      } catch (YellowException exc) {
         System.out.print("B-yellow,");
         throw new GreenException();
      } catch (RedException exc) {
         System.out.print("B-red,");
      } finally {
         System.out.print("B-finally,");
      }
      System.out.println("]B");
    }
```

What is Output When:

1. no exception is thrown

```
main[
A[
B[
```

2. `methodE` throws a `YellowException`?

```
main[
A[
B[
```

3. `methodC` throws a `GreenException`?

```
main[
A[
B[
```

4. `methodD` throws a `GreenException`?

```
main[
A[
B[
```

What is Output When:

5. `methodC` throws a `RedException`?

```
main[
A[
B[
```

6. `methodD` throws a `RedException`?

```
main[
A[
B[
```

7. `methodD` throws a `YellowException`?

```
main[
A[
B[
```

8. `methodD` throws a `OrangeException`?

```
main[
A[
B[
```

What is Output When:

9. `methodC` throws a `YellowException`?

```
main[
A[
B[
```

10. `methodC` throws a `BlueException`?

```
main[
A[
B[
```

11. `methodE` throws a `RedException`?

```
main[
A[
B[
```

What is Output When SOLUTION:

1. no exception is thrown

```
main[
A[
B[after C,after D,B-finally,]B
after B,]A
after A,after E,main-finally,]main
```

2. `methodE` throws a `YellowException`?

```
main[
A[
B[after C,after D,B-finally,]B
after B,]A
after A,main-finally,Exception in thread "main" YellowException
        at ExceptionTester.methodE(ExceptionTester.java:145)
        at ExceptionTester.main(ExceptionTester.java:37)
```

3. `methodC` throws a `GreenException`?

```
main[
A[
B[main-green,main-finally,]main
```

4. `methodD` throws a `GreenException`?

```
main[
A[
B[after C,B-finally,main-green,main-finally,]main
```

What is Output When SOLUTION:

5. `methodC` throws a `RedException`?

```
main[
A[
B[main-red,main-finally,]main
```

6. `methodD` throws a `RedException`?

```
main[
A[
B[after C,B-red,B-finally,]B
after B,]A
after A,after E,main-finally,]main
```

7. `methodD` throws a `YellowException`?

```
main[
A[
B[after C,B-yellow,B-finally,main-green,main-finally,]main
```

8. `methodD` throws a `OrangeException`?

```
main[
A[
B[after C,B-finally,main-finally,Exception in thread "main"
OrangeException
        at ExceptionTester.methodD(ExceptionTester.java:129)
        at ExceptionTester.methodB(ExceptionTester.java:80)
        at ExceptionTester.methodA(ExceptionTester.java:60)
        at ExceptionTester.main(ExceptionTester.java:34)
```

What is Output When SOLUTION:

9. `methodC` throws a `YellowException`?

```
main[
A[
B[main-finally,Exception in thread "main" YellowException
        at ExceptionTester.methodC(ExceptionTester.java:109)
        at ExceptionTester.methodB(ExceptionTester.java:76)
        at ExceptionTester.methodA(ExceptionTester.java:60)
        at ExceptionTester.main(ExceptionTester.java:34)
```

10. `methodC` throws a `BlueException`?

```
main[
A[
B[A-blue,]A
after A,after E,main-finally,]main
```

11. `methodE` throws a `RedException`?

```
main[
A[
B[after C,after D,B-finally,]B
after B,]A
after A,main-red,main-finally,]main
```