Class 13 Notes

First a little review:

In the last class we looked at the Wrapper classes:
Integer, Double, Boolean, Character,Float, Long, Byte

These allow us to have genuine objects corresponding to int, double,char, boolean.

Here is an example where this is convenient. We can sort an array of integers using our generic SelectionSort.sort(Comparable[] x, int n) method:

```
import java.util.*;
public class SortInteger
{
  public static void main(String[] args)
  {
    Random r = new Random();
    Integer[] x = new Integer[10];
    for (int i = 0; i < 10; i++)
    {
      int number = r.nextInt(100);
      x[i] = number;  // Autoboxing
    }
    SelectionSort.sort(x, 10);
    for(int i = 0; i < 10; i++)
      System.out.println(x[i]);
  }
}
```

Output:

```
7
10
10
34
36
37
46
52
60
96
```

New Stuff: Exceptions

An ***Exception*** is an abnormal condition that occurs at runtime.

Example:

- Null pointer Exception
- Array out of bounds Exception
- Type mismatch exception

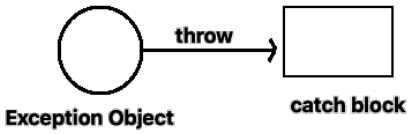
Java provides a mechanism to handle exceptions explicitly:
the try-catch-throw construction

Here is what happens when an exception occurs:

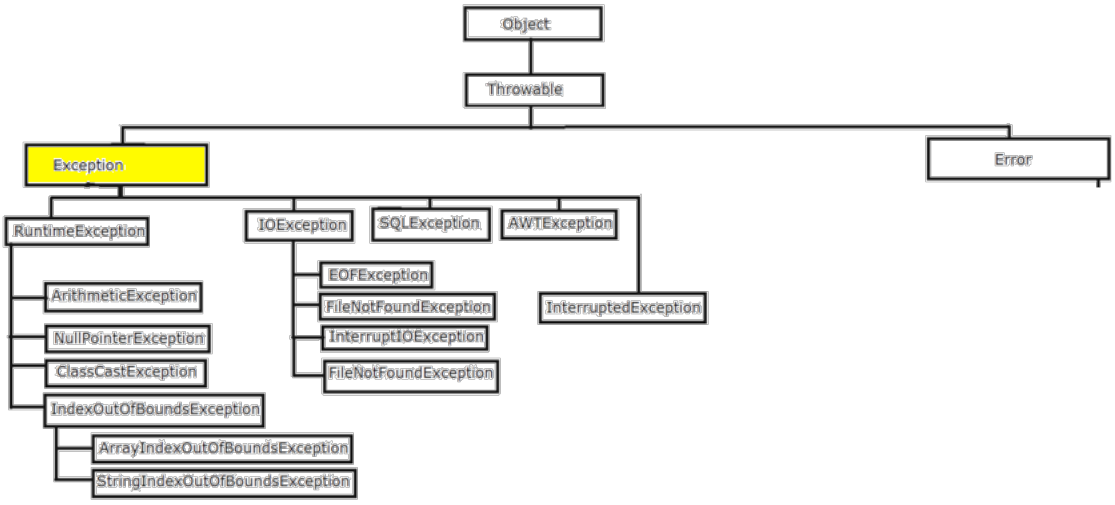
1. An Exception object is created. This is a genuine object and like all objects this holds information about the exception. The Exception object is created by the Java Virtual machine or the program.

2. The Exception object is passed ("thrown") to a "catch block." The catch block handles the exception. The catch block is a section of code.

3. The program continues with the code following the catch block.



Java has a hierarchy of Exception classes. Here is a partial picture of the Exception hierarchy:



These are all classes that **extend** Exception. So NullPointerException is-a Exception.

Here is a very, very simple example of how the try-catch- throw mechanism works. The program can obviously be written with a simple if-else but it will illustrate how the mechanism works.

```
public class BankAccount
{
     private int balance;
     public BankAccount()  //default constructor
    {
      balance = 0;
     }
     public BankAccount( int deposit) // one arg constructor
    {
      balance = deposit;
     }
     public void withdraw(int amount)
     {
       try
      {
        if(amount > balance)
        {
         Exception e = new Exception("Withdrawal exceeds balance");
         throw e;
        }
       balance = balance - amount;
      }
      catch (Exception e)
     {
         System.out.println(e.getMessage());
         System.out.println("You don't have enough money");
     }
     System.out.println("Current Balance : " + balance);
   }

    public void deposit(int deposit)
   {
     balance = balance+ deposit;
    }

     // getters and setters go here
}
```

```
import java.util.*;
public class TestBankAccount
{
   public static void main(String[] args)
   {
      Scanner input = new Scanner(System.in);
      BankAccount bank = new BankAccount(1000);
      String again = "";
     do
     {
       System.out.print("Enter withdrawal amount: ");
       int amount = input.nextInt();
       bank.withdraw(amount);
       System.out.println("Again -- Y for yes: ");
        again = input.next();
    }while (again.equals("Y"));
 }
}
----------Output----------
Enter withdrawal amount:  200
Current Balance : 800
Again -- Y for yes:  Y

Enter withdrawal amount:  300
Current Balance : 500
Again -- Y for yes:  Y

Enter withdrawal amount:  2000
Withdrawal exceeds balance
 You don't have enough money
Current Balance : 500
Again -- Y for yes:  Y

Enter withdrawal amount:  100
Current Balance : 400
Again -- Y for yes:  N
```

So what is happening here?

If the withdrawal exceeds the current balance

1. An Exception object e is created and initialized with the String **"Withdrawal exceeds balance"**
   This is done with the statement:
   Exception e = new Exception("Withdrawal exceeds balance");

2. The exception object, e, is thrown (or passed) to the catch block.
   throw e;
   Any additional statements in the try block (**balance = balance – amount**) are skipped

3. Control passes to the catch block. That is the red code

4. The getMessage() method is a method of the Exception class and just returns the string contained in e.

5. Program resumes with the code following the catch block (Green statement)

Format:

```
try
{
        code
        instantiate an Exception object. (It could have any name e, exp, harry)
        throw the exception object
        possibly more code
}
catch (Exception e)
{
        Code that handles the exception
}
```

You can instantiate the Exception object with a message, but that is not necessary.
Exception e = new Exception() is OK too.

In the Bank Account example, the program explicitly creates an Exception object and throws it.

```
Exception e = new Exception(" Withdrawal exceeds balance");
throw e;
```

Here is another example where the program handles the Exception

```
import java.util.*;
public class ProgramHandlesExceptions
{
    public static void main(String[] args)
    {
       Scanner input = new Scanner(System.in);
       boolean correct = false;
       while (!correct)
       {
        correct = true;
        String numberString ="";
        try
        {
           System.out.print("Enter a number ");
           numberString = input.next();
           for (int i = 0; i < numberString.length(); i++)
              if (!Character.isDigit(numberString.charAt(i)))
              {
                 Exception e = new Exception("Evil input "+ numberString + " reenter ");
                  throw e;
              }
        }
          catch (Exception e)
          {
            System.out.println(e.getMessage());
            correct = false;
          }
          if (correct)
          System.out.println(" You entered "+ numberString);
       } // end while
    }
}
```

Notice

- after executing the catch block the program continues. **That is, the loop continues**.

- numberString was declared outside the try block. Otherwise, it would be visible only in that block

It is often the case that the Java Virtual Machine ("The System") creates and throws the exception

For example:

public class SystemThrownException
{
public static void main(String[] args)
{
int[] x = new int[10];
x[1] = 5;
x[2] = 6;
x[13] = 7;
}
}
	java.lang.ArrayIndexOutOfBoundsException: 13
	at
	SystemThrownException.main(SystemThrownException.java:9)

In this case, the JVM made an ArrayIndexOutOfBoundsException and threw it.
And in this case, the system handled it with the ugly red error message.

So an Exception object can be thrown

1. **Explicitly by your program** as in the Bank Account program

2. **By the JVM.** If the JVM throws the Exception there is no explicit "throw statement."The SystemThrownException class is an example.

IMPORTANT

If a program, such as BankAccount, explicitly creates and throws an Exception. It is done in a try block and there must be a catch block to handle the exception

If the JVM throws the Exception and there is no catch block then the system will handle it usually by crashing the program and issuing an ugly error message.
However, a program can handle a system generated Exception with try-catch blocks
}

For the most part we will deal with System generated Exceptions and that can get a little complex.

System throws and catches an Exception. No try-catch is used.

```
import java.util.*;
public class Simple
{
 // System throws and catches the exception
 public static void main(String[] args)
 {
  Scanner input = new Scanner(System.in);
  System.out.print("Enter n: ");
  int n = input.nextInt();
 }
}
```

```
Enter n:  A
java.util.InputMismatchException
        at java.util.Scanner.throwFor(Scanner.java:840)
        at java.util.Scanner.next(Scanner.java:1461)
        at java.util.Scanner.nextInt(Scanner.java:2091)
        at java.util.Scanner.nextInt(Scanner.java:2050)
        at Simple.main(Simple.java:9)
>
```

Here is another example. For now ignore the "throws IOException clause." I'll explain that soon.

Here the JVM throws a FileNotFoundException and catches it.
The JVM handles the exception by issuing a message and terminating the program.

Notice: No try-catch

```
import java.util.*;
import java.io.*;
public class CatchMe1
{
   // Java throws and catches  the exception
   // Notice no "throws IOException"

 public static void main(String[] args) throws IOException
 {
  File f = new File("Fake1.txt");
  int n = 0;
  Scanner input = new Scanner(f);
  n = input.nextInt();
  System.out.println("The square is "+ (n*n));
 }
}
```

```
java.io.FileNotFoundException: Fake1.txt (No such file or directory)
        at java.io.FileInputStream.open(Native Method)
        at java.io.FileInputStream.<init>(FileInputStream.java:120)
        at java.util.Scanner.<init>(Scanner.java:636)
        at CatchMe1.main(CatchMe1.java:12)
```

- **Here the System throws an Exception and the program catches it.**
- **Notice there is no throw statement here.**
- **When the System throws an Exception the program does not create and throw an Exception object. The JVM does that.**

- **Here we need a try-catch so that our program can catch and handle the Exception**

```
import java.util.*;
import java.io.*;
public class CatchMeAgain
{
   // Java throws the exception, program catches it
   // Notice no "throws IOException"

 public static void main(String[] args)
 {
  File f = new File("realfile.txt");
  int sum = 0;

  try
  {
   Scanner in = new Scanner(f);  // exception here
   while( in.hasNext())
         sum = sum+ in.nextInt()
  }
  catch (Exception e)
  {
    System.out.println("Doh! Bad file: "+
                  f.getName()+ "   "+ e.getMessage());
    System.exit(0);
  }
  System.out.println("The sum is "+ sum);
 }
}
```

When run with a real file:
Output:

The sum is 25

When run with "fake.txt"

Doh! Bad file: fake.txt fake.txt (No such file or directory)

"fake.txt (No such file or directory)"

Is the resut of e.getMessage() –
The default mssage
"

Notice Integer.parseInt(String s) throws a NumberFormatException:

```
public class X
{
  public static void main(String[] args)
  {
    int a = Integer.parseInt("34a");// oops
    System.out.println(a);
  }
}
```

```
java.lang.NumberFormatException: For input string: "34a"
        at java.lang.NumberFormatException.forInputString(NumberFormatException.jav
        at java.lang.Integer.parseInt(Integer.java:458)
        at java.lang.Integer.parseInt(Integer.java:499)
        at X.main(X.java:5)
```

A better version of ReadData class that reads ints and doubles

```
import java.util.*;
public class ReadDataImproved
{
   public static int readInt()
   {
      Scanner input = new Scanner(System.in);
      boolean correct = false;         // is data correct?
      String number;          // input is a string
      int value = 0;
      while(! correct)        // until a correct values is entered
      {
        try
        {
           number = input.next();
           value = Integer.parseInt(number);     // NumberFormatException is possible here
           correct = true;                        // parseInt(number) had no problem
        }
        catch (NumberFormatException e)
        {
           System.out.println("Input error;  Re-enter: ");
        }
      }
      return value;
   }
```

```
  public static double readDouble()
  {
     // returns a valid double that is supplied interactively
     Scanner input = new Scanner(System.in);
     boolean correct = false;   // is data correct?
     String number;    // input string
     double value = 0.0;
     while(! correct)     // until a correct values is entered
     {
        try
        {
           number = input.next();
           value = Double.parseDouble(number);  // a possible exception
           correct = true;
        }
        catch (NumberFormatException e)
        {
           System.out.println("Input error; Re-enter: ");
        }
     }
     return value;
  }
}
```

Using ReadDataImproved:

```
public class TestReadData1
{
  public static void main(String[] args)
  {

    int sum = 0;
    for (int i = 0; i < 5; i++)
    {
      int num = ReadDataImproved.readInt();
      sum = sum + num;
    }

    System.out.println("Sum is "+ sum);
  }
}
```

```
Output:
1
 2
 3r
Input error;  Re-enter:
 3
4
5a
Input error;  Re-enter:
5
Sum is 15

Look! No crash
```