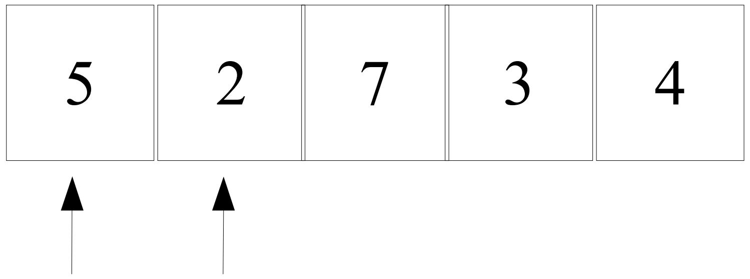
Bubble Sort

- Input: an array of elements (e.g. integers)
- Output: an array containing all the elements provided as input in sorted order
- Steps
  - take two adjacent elements at a time and compare them.
    - if input[i] > input[i+1]
      - swap
  - repeat until no more swaps are possible.

5	2	7	3	4

Bubble Sort

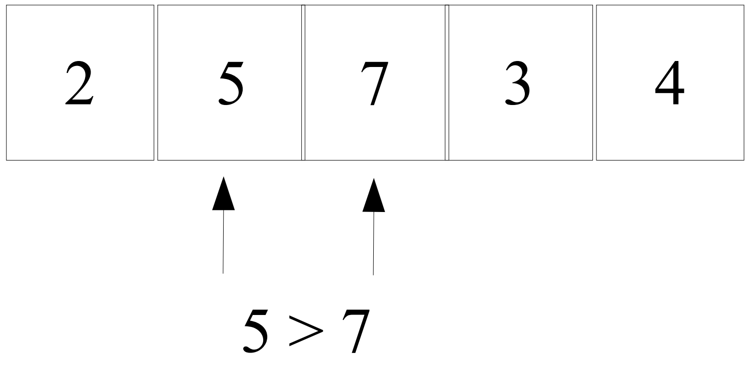
- Input: an array of elements (e.g. integers)
- Output: an array containing all the elements provided as input in sorted order
- Steps
  - take two adjacent elements at a time and compare them.
    - if input[i] > input[i+1]
      - swap
  - repeat until no more swaps are possible.



5 > 2 swap

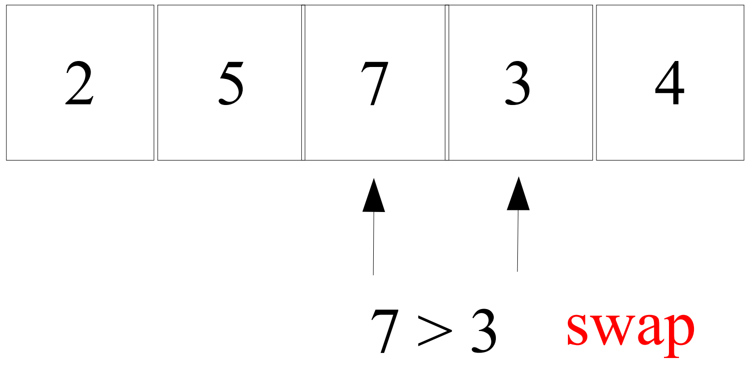
Bubble Sort

- Input: an array of elements (e.g. integers)
- Output: an array containing all the elements provided as input in sorted order
- Steps
  - take two adjacent elements at a time and compare them.
    - if input[i] > input[i+1]
      - swap
  - repeat until no more swaps are possible.



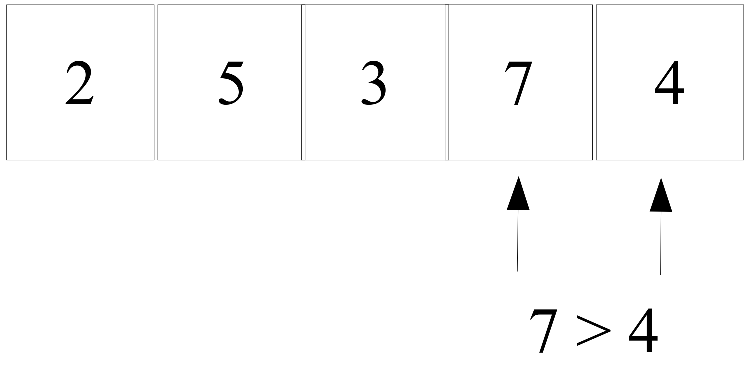
Bubble Sort

- Input: an array of elements (e.g. integers)
- Output: an array containing all the elements provided as input in sorted order
- Steps
  - take two adjacent elements at a time and compare them.
    - if input[i] > input[i+1]
      - swap
  - repeat until no more swaps are possible.



Bubble Sort

- Input: an array of elements (e.g. integers)
- Output: an array containing all the elements provided as input in sorted order
- Steps
  - take two adjacent elements at a time and compare them.
    - if input[i] > input[i+1]
      - swap
  - repeat until no more swaps are possible.



Bubble Sort

- Input: an array of elements (e.g. integers)
- Output: an array containing all the elements provided as input in sorted order
- Steps
  - take two adjacent elements at a time and compare them.
    - if input[i] > input[i+1]
      - swap
  - repeat until no more swaps are possible.

2	5	3	4	7

Repeat

Bubble Sort

```java
public class BSort {

  public static int[] sort(int[] input){
    int temp;
    boolean done = false;

    while(!done){
      done = true;
      for (int i=0; i<(input.length -1); i++){
        if(input[i] > input[i+1]){
          temp = input[i];
          input[i] = input[i+1];
          input[i+1] = temp;
          done=false;
        }
      }
    }
    return input;
  }
}
```

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

    int[] test1 = {10,3,6,7,2,4,9,5,1,8};
    test1 = BSort.sort(test1);
    System.out.println(
        Main.intArrayAsString(test1));
  }

  public static String
         intArrayAsString(int[] in){
    StringBuffer result = new StringBuffer();
    result.append("[ ");
    for (int i = 0 ; i < in.length ;i++){
      result.append(in[i]+" ");
    }
    result.append(" ]");
    return result.toString();
  }
}
```

`java.util.Collections` provide a variety of algorithms for sorting searching, shuffling etc.

Complexity of Bubble Sort

- Worst Case:
  - input is in reverse order. (10,9,8,7,6,5,4,3,2,1)
  - how many swaps
    - 9 + 8 + 7 + ... + 1
    - $\frac{(n-1)*n}{2}$
    - $O(n^2)$
- Best Case:
  - input is in order (1,2,3,4,5,6,7,8,9,10)
  - the algorithm still goes over each element once and checks if a swap is necessary.
  - n

Input/Output in Java (I/O)

- Look into `java.io.*`
- Information can be read from or written to a source.
  - information can be
    - human readable
    - machine readable
  - source/target can be
    - a file on your file system
    - a network connection (socket)
- Three roles are important for the whole setup
  - Program (your code)
  - The stream (the flow of information)
  - The source/target

Input/Output in Java (I/O) (cont)

- Writing (or reading) to (or from) a stream follows a simple template

Reading:

```
open a stream
while has more info
    get info
close stream
```

Writing:

```
open a stream
while more info
    write info
close stream
```

Input/Output in Java (I/O) (cont)

- Java provides the same operations but for 2 different data, *characters* and *bytes*
- Character Streams
  - sub-divided into
    - `Writer`: abstract class that defines the API and partial implementation for writers
    - `Reader`: abstract class that defines the API and partial implementation for readers
  - Sub-classes of `Reader`,`Writer` are specializations on different kinds of streams,
    - Character, string, buffered, piped etc.

Input/Output in Java (I/O) (cont)

- Java provides the same operations but for 2 different data, *characters* and *bytes*
- Byte Streams
  - sub-divided into
    - `InputStream`: abstract class that defines the API and partial implementation for reading bytes
    - `OutputStream`: abstact class that defines the API and partial implementation for writing bytes
  - Subclasses of `InputStream` and `OutputStream` are specializations on different kinds of streams,
    - FileStream, ObjectInputStream, ByteArrayOutputStream etc.

Working with Files

- Java provides a class that represents a file on your file system. java.io.File.
- An example is the copying a file.

```
import java.io.*;
public class Copy {
    public static void main(String[] args) throws IOException
{
        File inputFile = new File("farrago.txt");
        File outputFile = new File("outagain.txt");

        FileReader in = new FileReader(inputFile);

        FileWriter out = new FileWriter(outputFile);

        int c;
        while ((c = in.read()) != -1)
           out.write(c);
        in.close();
        out.close();
    }
}
```

Represent the file "farrago.txt" and "outagain.txt" as Java objects

Wrap a FileReader around the input file

Wrap a FileWriter around the output file

Digression, Java Exceptions

- Exceptions are Java's way of dealing with error-handling
- Why exceptions?
  - Separates error handling code from “regular code”
  - propagate errors
  - group error types together and error differentiation
- Using Java lingo:
  - exceptions are **thrown** when something out of the ordinary happens
  - exceptions can be **caught** in order to provide code for error recovery or more informative error messages.

Java syntax for exceptions (throwing)

- Declare a method that throws exceptions

```
public class Stack{

 public Object pop() throws StackEmptyException {
   if (isEmpty())
     throw new StackEmptyException();
   else
      return this.getFirstElement();
 }
}
```

- Java's throw statement takes an object instance of type *Throwable.*

Java syntax for exceptions (try-catch)

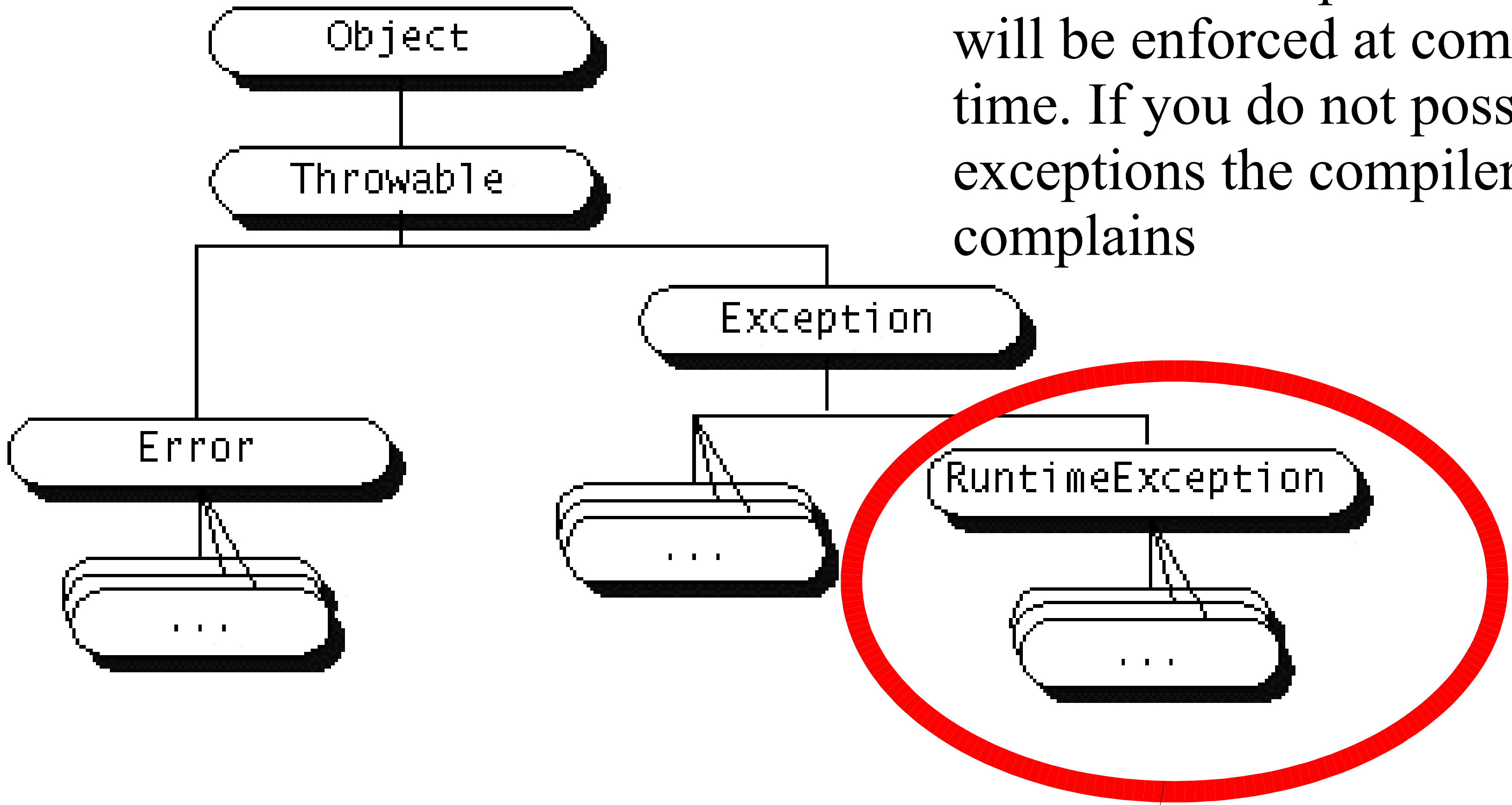
- try-block allows you to execute code and in the case of an exception, then that will be caught.
- catch-block will only execute once the exception has been thrown.

```
public class Main{

 public static void main(String[] args){
  Stack s = new Stack();
  try{
     Object result = s.pop();
   } catch (StackEmptyException empty){
     System.out.println(empty.getMessage());
   } catch (ArrayIndexOutOfBounds bounds){
     System.out.println(bounds.getMessage());
   }
}
```

Java Exceptions are Objects!

- Checked exceptions will be enforced at compile time. If you do not possible exceptions the compiler complains



Java does not force you to deal with these exceptions explicitly in your code

Java Exceptions

```
import java.io.*;
public class Copy {
    public static void main(String[] args)
//throws IOException
{
   File inputFile = new File("farrago.txt");
   File outputFile = new File("outagain.txt");

   FileReader in = new FileReader(inputFile);

   FileWriter out = new FileWriter(outputFile);

   int c;
   while ((c = in.read()) != -1)
      out.write(c);
   in.close();
   out.close();
    }
}
```

- remove “throws” part
- Does not compile
  - `in.read` is defined to throw an exception and this code does not deal with it.
- You **have** to take care of possible exceptions
  - *except runtime exceptions*
- You can
  - propagate the exception up
  - deal with it in your code

Java Exceptions. Pass it on.

```
import java.io.*;
public class Copy {
    public static void main(String[] args)
throws IOException
{
   File inputFile = new File("farrago.txt");
   File outputFile = new File("outagain.txt");

   FileReader in = new FileReader(inputFile);

   FileWriter out = new FileWriter(outputFile);

   int c;
   while ((c = in.read()) != -1)
      out.write(c);
   in.close();
   out.close();
    }
}
```

- Make sure that the whole method propagates any exception thrown in the method body
  - you are postponing error handling by passing the exception to the method that called you.
- In this case the JVM !
  - simply prints out the message that comes along with the exception.

Java Exceptions. Deal with it now.

```
import java.io.*;

public class Copy {
    public static void main(String[] args)
{
try{
   File inputFile = new File("farrago.txt");
   File outputFile = new File("outagain.txt");

   FileReader in = new FileReader(inputFile);

   FileWriter out = new FileWriter(outputFile);

   int c;
   while ((c = in.read()) != -1)
      out.write(c);
   in.close();
   out.close();

  } catch (IOException ioException){

    System.out.println(ioException.getMessage();
    }
   }
}
```

Try to execute this code

If something went wrong, then it will throw a **known** exception. Catch that object and give it a name so that it can be used in the *catch* block

Code that has access to the instance of the exception thrown. Deal with the exception accordingly

Last note on I/O

- The copy application reads in a character at a time
  - BufferedReader/BufferedWriter are provided for easier manipulation
  - Simply wrap them around your input/output stream

```
import java.io.*;
public class Copy {
    public static void main(String[] args)
throws IOException
{
   File inputFile = new File("farrago.txt");

   FileReader in = new FileReader(inputFile);
   BufferedReader bRead = new BufferedReader(in);

   while (bRead.ready())
      System.out.print(bRead.getLine());
   in.close();
    }
}
```

Getting information from the System

- Java allows you to view, add and alter system properties
  - these name, value pairs that hold system specific information (i.e. execution path)

```
import java.util.*;
import java.io.*;

public class Main {

  public static void main(String[] args){
    Properties mySystemProps = System.getProperties();

    PrintStream pStream = new PrintStream(System.out);
    mySystemProps.list(pStream);
    System.out.println("*****");
    System.out.println("");
  }
}
```

Executing System commands

- You can use Java to call system specific commands
  - you can combine commands and process their output

```java
import java.util.*;
import java.io.*;

public class Main {

  public static void main(String[] args){
    Runtime runTime = Runtime.getRuntime();
    try{
      Process ls = runTime.exec("ls");
      ls.waitFor();
      System.out.println("Output from ls ...\n");
      InputStream iStream = ls.getInputStream();
      InputStreamReader iSReader = new InputStreamReader(iStream);
      BufferedReader bfReader = new BufferedReader(iSReader);
      while (bfReader.ready())
        System.out.print(bfReader.readLine()+"\n");
    }catch (Exception exc){
      exc.printStackTrace();
    }
  }
}
```