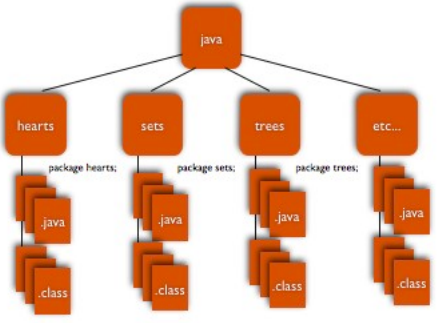
Java Basics

Packages, Streams, Exceptions

The Base Directory and Packages

- Choose a base directory that will be the root of all of your java work – mine is /Users/apasik/src/java.
- All of your work (edit, compile, run) must be done from this base directory.
- Each project or application you build should be within a subdirectory of the base directory – this is also called the package.

Base Directory and Packages



```
% cd java
% ls
hearts      sets        trees
% javac sets/*.java
% java sets/MakeSet 4 2 9 7 9 5 2 3
{ 3 5 7 9 2 4 }
%
```

Packages

- All source files (.java) in one package (i.e., one subdirectory) must start with the line

  package <package-name>;

- All public names (classes, methods, static variables, and instance variables) within a package must be unique, but may be non-unique with names in another package.
- If code in one package needs access to classes in another package, that code must import those classes with the line

  import <other-package>.*;

Streams

- Text-based input and output are handled through streams.
- The PrintStream class is for output, and the BufferedReader class is for input.
- The io package (posted on courseworks) with the single final class IO provides all the basic input and output methods you need.

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

public final class IO {

    public static BufferedReader stdin =
        new BufferedReader(new InputStreamReader(System.in));
    public static PrintStream stdout = System.out;
    public static PrintStream stderr = System.err;
.
.
.
}
```

```
public final class IO {
.
.
.
    public static String readFile (String name) {
        String s = "", line;
        BufferedReader f;
        try {
            f = new BufferedReader(new FileReader(name));
            while ((line = f.readLine()) != null)
                s += line + "\n";
            f.close();
        }
        catch (IOException e) {
            stderr.println("Can't open file: " + name);
        }
        return s;
    }
.
.
.
}
```

```
public final class IO {
.
.
.
    public static String prompt (String s) {
        try {
            stdout.print(s);
            stdout.flush();
            return stdin.readLine();
        }
        catch (IOException e) {
            stderr.println(e); return "";
        }
    }
.
.
.
}
```

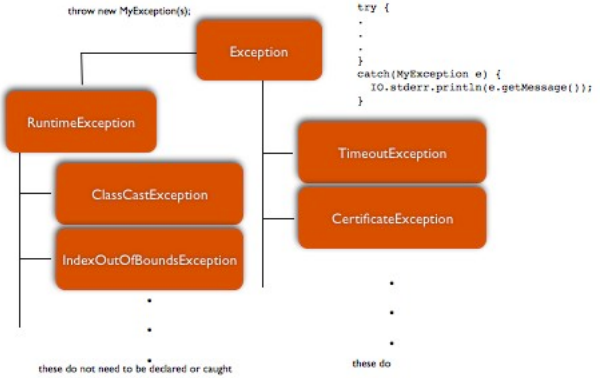
```
public final class IO {
.
.
.
    public static int promptInt (String s, int lo, int hi) {
        try {
            stdout.print(s + " [" + lo + " to " + hi + "] ");
            stdout.flush();
            int result = Integer.parseInt(stdin.readLine());
            if (result < lo || result > hi)
                throw new IOException("Out of range.");
            return result;
        }
        catch (Exception e) {
            stderr.println(e); return lo - 1;
        }
    }
.
.
.
}
```

```
public final class IO {
.
.
.
    public static boolean affirmative (String yn) {
        return (yn.charAt(0) == 'y') || (yn.charAt(0) == 'Y');
    }
.
.
.
}
```

Exceptions

throw new MyException(s);

```
try {
.
.
.
}
catch(MyException e) {
  IO.stderr.println(e.getMessage());
}
```



these do not need to be declared or caught

these do

```
public int f (MyObj x) throws MyEx {
.
.
.
    if (x == null)
       throw new MyRTEx("Null arg to f not allowed.");
    if (x.intPart == 0)
       throw new MyEx("intPart of arg to f cannot be 0.");
.
.
.
}

public int g (MyObj y) {
.
.
.
try { int z = f(y); }
catch (MyEx e) { IO.stderr.println(e.getMessage()); }
.
.
.
}
```

Why RuntimeExceptions?

"Runtime exceptions represent problems that are the result of a programming problem, and as such, the API client code cannot reasonably be expected to recover from them or to handle them in any way. Such problems include arithmetic exceptions, such as dividing by zero; pointer exceptions, such as trying to access an object through a null reference; and indexing exceptions, such as attempting to access an array element through an index that is too large or too small."