Review: Basic types

Built-in types	`int, byte, short, long`
	`float, double`
	`boolean`
	`String ("abcd")`
	`char ('c')`

Arithmetic operators	`+, -, *, /` (overloaded) `%`
Comparisons	`>, >=, <, <=` `==, !=`
Boolean Operators	`&&, ||, not`

Review: Arrays

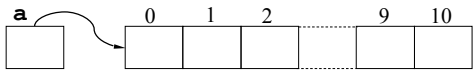
Declaration

```
int[] a = new int[10];
```

Access

```
for (int i=0; i<a.length; i++)
    System.out.println(a[i]);
```

The variable **a** is a reference



A simple Java program

```
public class E1 {
  public static void main(String[] arg){
     int x = 2;
     int y = x + 3;
     System.out.println("y+4");
  }
}
```

What does this print?

"Static"

```
public class E0
{
   public static void main(String[] args)
   {
   }
}
```

"static" means that you can call this method without an instance of the class

Uses of static methods:

Math.random() (and lots of other math functions)
"main" functions of classes
Integer.parseInt(), etc

Command line arguments

`public static void main (String[] arg)`

The array **`arg`** is an array of strings

Each element of this array is an argument given to your program, if any.

```
public class E2 {
    public static void main(String[] arg)
    {
  for (int i=0; i<arg.length; i++)
      System.out.println(i + ": " + arg[i]);
    }
}
```

References and values

Java has two kinds of variables:

"by reference" -- all arrays and upper case names (String)
"by value" -- all lower case names (int)

- By reference
  - variables holds the place in memory where the value is stored
  - assignments copy the memory location
  - equality tests for the same memory location
- By Value
  - variables hold values themselves
  - assignments copy values
  - equality tests for the same value

References and Values -- Example

```
public class E11
{
   public static void main(String[] args)
   {
        int[] a=new int[1];
        a[0]=1;
        int[] b=a;
        b[0]=6;
        System.out.println(a[0] + " " + b[0] + " " + (a[0]==b[0]));

        int i=5;
        int j=i;
        i=7;
        System.out.println(i + " " + j + " " + (i==j));
   }
}
```

Exceptions

Exceptions are Java's way of telling the programmer that something is wrong.

A method will ***throw*** a corresponding exception if an error is detected

Some exceptions must be "caught", some can just happen

Compiler will not succeed when you must catch an exception

Handling the unexpected

An exception may be caught, or passed on.

Catching an exception means handling it:

```
try {
// some code that could throw the
named exception
} catch (someException e) {
// code to deal with the error
}
```

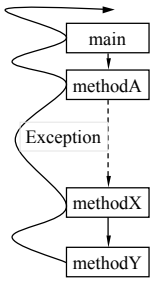
`try` and `catch`

```
public class E21
{
   public static void main(String[] args){
    try {
       for (int i=0; i<args.length; i++)
           {
             int num = Integer.parseInt(args[i]);
             System.out.println(i + " arguement is an integer");
           }
    } catch (NumberFormatException e) {
       System.out.println("A number was expected");
    }
   }
}
```

Question: is this a good program? How could it be better?

Passing it on

Uncaught exceptions are passed on to the calling method through the invocation chain



`throws`

Every method that throws an exception must acknowledge it in its declaration

e.g. public void a() throws Exception

Every method that uses a method that throws an exception must either catch the exception in its body or throw the exception in its declaration

There are lots of kinds of exceptions -- always good to be specific but often boring and unnecessary

catch and throw

```
public class E3
{
   public static void main(String[] args)
   {
         try
            {
                  parseArgs(args);
            }
         catch (Exception ee)
            {
                  System.err.println("An error!!!");
            }
   }
   public static void parseArgs(String[] args) throws Exception
   {
         for (int i=0; i<args.length; i++)
                  System.out.println(17 / Integer.parseInt(args[i]));
          }
}
```

Specificity is good

```
public class E3a
{
   public static void main(String[] args)
   {
         try {
                  divArgs(args);
            }
         catch (ArithmeticException ee) {
                  System.err.println("Zero");
            }
         catch (NumberFormatException ee) {
                  System.err.println("Not an integer");
            }
   }
   public static void divArgs(String[] args) throws NumberFormatException,ArithmeticE
   {
         for (int i=0; i<args.length; i++)
            System.out.println(17 / Integer.parseInt(args[i]));
   }
}
```