

Java Programs

- You have to create a class!
- You run the class.
 - for this to work, the class must have a method named `main()` that is declared as

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

- if you don't do this right, you see something like this when you try to run a class:

```
Exception in thread "main" java.lang.NoSuchMethodError: main
```

`public static void main`

- **public**: This method can be accessed (called) from outside the class.
- **static**: This method does not require that an object of the class exists. static methods are sort-of like "global" methods, they are always available (but you have to use the class name to get at them).
- **void**: no return value.

`main (String[] args)`

- `main ()` will be passed an array of Strings corresponding to any command line parameter values.
- If you ran a program (class) like this:

```
java Foo hi there dave
```

- then `Foo.main ()` would be passed an array (with length 3) of String objects. The values would be "hi", "there" and "dave".
 - `args [0]` is the String "hi", ...

Program: Show Command Line

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

String concatenation

Converting **String** to **int**

```
int x = Integer.parseInt("12");
```

Integer is actually the name of a *class*.

parseInt() is one of the methods of class **Integer**.

The details are in the Java API documentation in **java.lang**, class **Integer**

Printing the command line args as `ints`

```
public class ShowArgs {  
    public static void main(String[] args) {  
        for (int i=0;i<args.length;i++)  
            System.out.println("args["+i+"] = "  
                + Integer.parseInt(args[i]));  
    }  
}
```

Issues

- What happens if the user doesn't type in a bunch of integer on the command line?

```
java ShowArgs hello there
```

- **Integer.parseInt** throws an exception.
 - we didn't catch the exception, the program is terminated and some default exception handling code is executed (a stack trace is printed).

Catching the Exception

```
public class ShowArgs {
    public static void main(String[] args) {
        try {
            for (int i=0;i<args.length;i++)
                System.out.println("args["+i+"] = "
                    + Integer.parseInt(args[i]));
        } catch (NumberFormatException e) {
            System.out.println("Error: I need ints!");
        }
    }
}
```

Another Method (better?)

```
public class ShowArgs {
    public static void main(String[] args) {
        for (int i=0;i<args.length;i++)
            try {
                System.out.println("args["+i+"] = "
                    + Integer.parseInt(args[i]));
            } catch (NumberFormatException e) {
                System.out.println("Skipping (not an int)");
            }
    }
}
```

Printing the *stack trace*

```
for (int i=0;i<args.length;i++)
    try {
        System.out.println("args["+i+"] = "
            + Integer.parseInt(args[i]));
    } catch (NumberFormatException e) {
        e.printStackTrace();
    }
```

Sample Stack Trace Output

```
java ShowArgsST 1 2 hello
args[0] = 1
args[1] = 2
java.lang.NumberFormatException: forInputString: "hello"
    at
    java.lang.NumberFormatException.forInputString(Number
    FormatException.java:48)
        at java.lang.Integer.parseInt(Integer.java:447)
        at java.lang.Integer.parseInt(Integer.java:497)
        at ShowArgsST.main(ShowArgsST.java:16)
```