public class Methods
{
    public static void main(String[] args)
    {
        methods();
        are();
        awesome();
    }
}
Topic 06: Methods

- Functions and Procedures 5.1
- Declaring a method 5.2
- Calling a method 5.2, 5.4, 5.5
- Methods calling methods
- Throwing Exceptions 7.4.1
Functions and Procedures

• Two views of a method
  - Method as a mathematical function
  - Method as a procedure
Mathematical Functions

\[ a = f(x, y) \]

- Output
- Name
- Inputs
Mathematical Functions

\[ a = f(x, y) \]

\[ f(x, y) = \sqrt{x^2 + y^2} \]
Mathematical Functions

\[ f(x, y) = \sqrt{x^2 + y^2} \]

\[ f(x, y) := \text{return } \sqrt{x^2 + y^2} \]

This is pseudocode.

Note the colon to show that this is a method, not a statement.
Mathematical Functions

\[ f(x, y) = \sqrt{x^2 + y^2} \]

public static double f(double x, double y) {
    return Math.sqrt(x*x + y*y);
}
Mathematical Functions

\[ f(x, y) = \sqrt{x^2 + y^2} \]

```java
public static double f(double x, double y) {
    return Math.sqrt(x*x + y*y);
}
```
public static double f(double x, double y)
{
    return Math.sqrt(x*x + y*y);
}

$f(x, y) = \sqrt{x^2 + y^2}$
public static double f(double x, double y) {
    return Math.sqrt(x*x + y*y);
}
Mathematical Functions

\[ f(x, y) = \sqrt{x^2 + y^2} \]

```java
public static double f(double x, double y) {
    return Math.sqrt(x * x + y * y);
}
```
Mathematical Functions

\[ f(x, y) = \sqrt{x^2 + y^2} \]

```java
public static double f(double x, double y) {
    return Math.sqrt(x*x + y*y);
}
```
Mathematical Functions Need Code!

```java
public static int min(int[] values) {
    ...
    return ...
}
```

**Group Activity:**
In the Arrays deck, we wrote code that would calculate the minimum value in an array.

Let's do that as a method.
Mathematical Functions Need Code!

```java
public static int min(int[] values) {
    int retval = values[0];

    for (int i=1; i<values.length; i++) {
        if (values[i] < retval) {
            retval = values[i];
        }
    }

    return retval;
}
```
Examples We've Seen of Functions

- `parseInt()`
- `Scanner.hasNext()`
public static void printX(int count)
{
    for (int i=0; i<count; i++)
    {
        System.out.print("X");
    }
}
Examples We've Seen of Procedures

- println()
- main()
Procedures

public static void printX(int count) {
    for (int i = 0; i < count; i++)
        System.out.print("X");
}

Most procedures have some input.
Procedures

```java
public static void printX(int count) {
    for (int i=0; i<count; i++)
        System.out.print("X");
}
```

Some procedures return nothing.

But many others return something.
Procedures

```java
public static void printX(int count) {
    for (int i=0; i<count; i++)
        System.out.print("X");
}
```

Procedures often affect the state of the world.
Procedures

public static void printX(int count)
{
    for (int i=0; i<count; i++)
        System.out.print("X");
}

return; is optional in a method that returns void.
Real World Methods

• Most methods are hybrids
  – Part function
  – Part procedure

• Do some calculation
• Do some work
• Return information about what happened
Topic 06: Methods

- Functions and Procedures 5.1
- Declaring a method 5.2
- Calling a method 5.2, 5.4, 5.5
- Methods calling methods
- Throwing Exceptions 7.4.1
Declaring a Method

Cut and Paste
Replace the blue stuff!

```java
public static int min(int[] values) {
    ... lots of code ... 
}
```
Declaring a Method

Method Name
Must be an identifier.

Convention is same as variables: lower case, then camel case.

```java
public static int min(int[] values) {
    ... lots of code ...
}
```
Declaring a Method

**Arguments**

Zero, one, or many variables. Give a type and name for each (like variable declarations). Separate with commas.

Also known as “parameters.”

```java
public static int min(int[] values)
{
    ... lots of code ...
}
```
Declaring a Method

Return Type
Exactly one type (with no name). Represents the output of the method. Use `void` to indicate that you will not return values.

```java
public static int min(int[] values)
{
    ... lots of code ...
}
```
Declaring a Method

**Body**

Do whatever you want!

*void* methods must never return any value. *Non-void* methods must return a value on every possible path.

```java
public static int min(int[] values) {
    ... lots of code ...
}
```
Declaring a Method

**public**
Optional. *Non-public* methods cannot be called by certain types of other code.

We'll use **public** for all our methods this semester. Next semester, that will change.

```
public static int min(int[] values)
{
    ... lots of code ...
}
```
Declaring a Method

```
public static int min(int[] values)
{
    ... lots of code ... 
}
```

**static**
Optional. Non-**static** methods can only be called once you have created an object of this class.

We'll cover this later this semester.
The Most Famous Method

```java
public static void main(String[] args) {
    ... lots of code ... 
}
```
Mixed Argument Types are OK

public static int[]
    resizeArray(int[] orig, int new_length)
{
    ... code ...
}

Arguments can have mixed types.
public static double f(double x, double y) {
    return Math.sqrt(x*x + y*y);
}

When declaring multiple arguments, every argument needs its own type … even if they are are are the same!
Remember this?

\[ f(x, y) = \sqrt{x^2 + y^2} \]

```java
public static double f(double x, y)
{
    return Math.sqrt(x*x + y*y);
}
```

**ILLEGAL!**

This code will not compile.
## Topic 06: Methods

- Functions and Procedures 5.1
- Declaring a method 5.2
- **Calling a method** 5.2, 5.4, 5.5
- Methods calling methods
- Throwing Exceptions 7.4.1
Calling a Method

\[ a = f(x, y) \]

```c
double a = f(1, 2);
```
Calling a Method

double a = f(1, 2);

**Method name**
Tells Java what method we want to call.
Calling a Method

double a = f(1,2);
Calling a Method

double a = f(1,2);
Calling a Method

double a = f(1,2);
Calling a Method

double a = f(1,2);

public static double f(int a, int b) {
    return a + b;
}
double a = f(1, 2);

public static double f(int a, int b) {
    return a + b;
}
Calling a Method

double a = f(1, 2);

public static double f(int a, int b)
{
    return a + b;
}
Calling a Method

double a = f(1, 2);

public static double f(int a, int b)
{
    return a + b;
}
Variants: Discard the Return Value

\[
f(1, 2);\]

Return Value Ignored
The return value is forgotten. It is lost!
Calling a \texttt{void} Function

```java
public static \texttt{void} g(int x, int y)
{
    ... 
}
```

g(1, 2);

\texttt{g( )} \textbf{returns nothing}
When you call \texttt{g( )}, it returns nothing.
Calling a `void` Function

```java
public static void g(int x, int y)
{
    // ...
}

int a = g(1, 2);
```

**ILLEGAL**
Your code will not compile.
g() returns void.
Topic 06: Methods

- Functions and Procedures 5.1
- Declaring a method 5.2
- Calling a method 5.2, 5.4, 5.5
- Methods calling methods
- Throwing Exceptions 7.4.1
Methods Calling Methods

- Most methods will call other methods
- Nesting again!
  - Build complex things from simple things
public static double $f(x, y) = \sqrt{x^2 + y^2}$

Remember this?

$$f(x, y) = \sqrt{x^2 + y^2}$$

f() calls Math.sqrt()
The Java Standard Library

• Java Standard Library is **HUGE**
  - Lots of good stuff!
  - http://docs.oracle.com/javase/7/docs/api/

• **In real world**, you should use it all the time!
The Java Standard Library

- Java Standard Library is **HUGE**
  - Lots of good stuff!
  - [http://docs.oracle.com/javase/7/docs/api/](http://docs.oracle.com/javase/7/docs/api/)

- In this class, please use only the things I've pointed out
  
  ```java
  Integer.parseInt()   Math.sqrt()
  println() / print()
  
  ... more to come ...
  ```
public static void printX(int count) {
    for (int i=0; i<count; i++)
        System.out.print("X");
}

printX() calls System.out.print()
public class MethodsCallingMethods1
{
    public static void bar()
    {
        System.out.println("we're in bar()");
    }

    public static void main(String[] args)
    {
        System.out.println("main: before foo()");
        foo();
        System.out.println("main: after foo()");
    }

    public static void foo()
    {
        System.out.println("foo: before bar()");
        bar();
        System.out.println("foo: after bar()");
    }
}
public class MethodsCallingMethods {
    public static void bar() {
        System.out.println("we're in bar()")
    }

    public static void main(String[] args) {
        System.out.println("main: before foo()")
        foo();
        System.out.println("main: after foo()")
    }

    public static void foo() {
        System.out.println("foo: before bar()")
        bar();
        System.out.println("foo: after bar()")
    }
}
public class MethodsCallingMethods {
    public static void bar() {
        System.out.println(“we’re in bar()”);
    }

    public static void main(String[] args) {
        System.out.println(“main: before foo()”);
        foo();
        System.out.println(“main: after foo()”);
    }

    public static void foo() {
        System.out.println(“foo: before bar()”);
        bar();
        System.out.println(“foo: after bar()”);
    }
}
public class MethodsCallingMethods {
    public static void bar() {
        System.out.println("we'");
    }

    public static void main(String[] args) {
        System.out.println("main: before foo()”);
        foo();
        System.out.println("main: after foo()”);
    }

    public static void foo() {
        System.out.println("foo: before bar()”);
        bar();
        System.out.println("foo: after bar()”);
    }
}
public class MethodsCallingMethods
{
    public static void bar()
    {
        System.out.println("we'
    }

    public static void main(String[] args)
    {
        System.out.println("main: before foo()"");
        foo();
        System.out.println("main: after foo()");
    }

    public static void foo()
    {
        System.out.println("foo: before bar()");
        bar();
        System.out.println("foo: after bar()");
    }
}
public class MethodsCallingMethods {
    
    public static void bar() {
        System.out.println("we're in bar()" );
    }

    public static void main(String[] args) {
        System.out.println("main: before foo() ");
        foo();
        System.out.println("main: after foo() ");
    }

    public static void foo() {
        System.out.println("foo: before bar() ");
        bar();
        System.out.println("foo: after bar() ");
    }
}
public class MethodsCallingMethods {
    public static void bar() {
        System.out.println("we're in bar()".}
    }

    public static void main(String[] args) {
        System.out.println("main: before foo()");
        foo();
        System.out.println("main: after foo()");
    }

    public static void foo() {
        System.out.println("foo: before bar()");
        bar();
        System.out.println("foo: after bar()");
    }
}
public class MethodsCallingMethods {
    
    public static void bar() {
        System.out.println("we're in bar()"); 
    }

    public static void main(String[] args) {
        System.out.println("main: before foo()"); 
        foo();
        System.out.println("main: after foo()"); 
    }

    public static void foo() {
        System.out.println("foo: before bar()"); 
        bar();
        System.out.println("foo: after bar()"); 
    }
}

Output:
main: before foo()
foo: before bar()
we're in bar()
foo: after bar()
public class MethodsCallingMethods {
    public static void bar() {
        System.out.println("we're in bar()\n        \n    }

    public static void main(String[] args) {
        System.out.println("main: before foo()")
        foo();
        System.out.println("main: after foo()")
    }

    public static void foo() {
        System.out.println("foo: before bar()")
        bar();
        System.out.println("foo: after bar()")
    }
}

Output:
main: before foo()
foo: before bar()
we're in bar()
foo: after bar()
main: after foo()
```java
public class MethodOfMethod {
    public static void main(String[] args) {
        int a = f(10);
        int b = f(a);
        int c = f(f(100));

        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println(f(f(1000)));
    }

    public static int f(int x) {
        return x + 1;
    }
}
```

Output:
```java
11
12
13
13
```
public class MethodOfMethod
{
    public static void main(String[] args)
    {
        int a = f(10);
        int b = f(a);
        int c = f(f(100));

        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println(f(f(1000)));
    }

    public static int f(int x)
    {
        return x+1;
    }
}

Output:
11
12
102
1002

In what order were these methods called?
public class MethodOfMethod
{
    public static void main(String[] args)
    {
        int a = f(10);
        int b = f(a);
        int c = f(f(100));

        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println(f(f(1000)));
    }

    public static int f(int x)
    {
        return x+1;
    }
}

Output:
11
12
102
1002

f(10) returns 11
public class MethodOfMethod
{
    public static void main(String[] args)
    {
        int a = f(10);
        int b = f(a);
        int c = f(f(100));

        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println(f(f(1000)));
    }

    public static int f(int x)
    {
        return x+1;
    }
}
public class MethodOfMethod {

    public static void main(String[] args) {
        int a = f(10);
        int b = f(a);
        int c = f(f(100));

        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println(f(f(1000)));
    }

    public static int f(int x) {
        return x + 1;
    }

}
public class MethodOfMethod {

    public static void main(String[] args) {
        int a = f(10);
        int b = f(a);
        int c = f(f(100));

        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println(f(f(1000)));
    }

    public static int f(int x) {
        return x + 1;
    }
}

Output:
11
12
102
1002

f(101) returns 102
public class MethodOfMethod {
    public static void main(String[] args) {
        int a = f(10);
        int b = f(a);
        int c = f(f(100));
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println(f(f(1000)));  
    }

    public static int f(int x) {
        return x+1;
    }

}

Output:
11
12
102
1002

Print out some values...
public class MethodOfMethod {
    public static void main(String[] args) {
        int a = f(10);
        int b = f(a);
        int c = f(f(100));
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println(f(f(1000)));
    }
    public static int f(int x) {
        return x + 1;
    }
}

Output:
11
12
102
1002

f(1000) returns 1001
public class MethodOfMethod
{
    public static void main(String[] args)
    {
        int a = f(10);
        int b = f(a);
        int c = f(f(100));

        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println(f(f(1000)));
    }

    public static int f(int x)
    {
        return x+1;
    }
}
public class MethodOfMethod
{
    public static void main(String[] args)
    {
        int a = f(10);
        int b = f(a);
        int c = f(f(100));

        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println(f(f(1000)));
    }

    public static int f(int x)
    {
        return x+1;
    }
}

Output:
11
12
102
1002

Print out 1002
Topic 06: Methods

- Functions and Procedures 5.1
- Declaring a method 5.2
- Calling a method 5.2, 5.4, 5.5
- Methods calling methods
- Throwing Exceptions 7.4.1
Exceptions

- A way to signal major errors
- Terminate a method immediately, like `return`
  - Does **NOT** return anything
  - No matter the expected return type
- Terminates calling method, too (unless caught)
- Eventually kills entire program (unless caught)
Standard Exception Types

- **ArrayIndexOutOfBoundsException**
  - You used an invalid array index
- **NumberFormatException**
  - Thrown by parseInt() if the String isn't an integer
- **IllegalArgumentException**
  - If the arguments to a method are not allowed

... and many more ...
How to Throw an Exception

throw new <type>();

- Creates a new object
- Throws it

Don't forget the parens and the semicolon!
How to Throw an Exception

throw new IllegalArgumentException();

throw new ArrayIndexOutOfBoundsException();

throw new NullPointerException();
public static int min(int[] values) {
    int retval = values[0];

    for (int i=1; i<values.length; i++) {
        if (values[i] < retval) {
            retval = values[i];
        }
    }

    return retval;
}
public static int min(int[] values)
{
    int retval = values[0];

    for (int i=1; i<values.length; i++)
    {
        if (values[i] < retval)
            retval = values[i];
    }

    return retval;
}

Throwing Exceptions

This code accesses an element without checking the length.

What if the array was empty?
The new code checks for the error case and throws an exception.
Topic 06: Methods

- Functions and Procedures 5.1
- Declaring a method 5.2
- Calling a method 5.2, 5.4, 5.5
- Methods calling methods
- Throwing Exceptions 7.4.1

Summary