Java Basics

Assumptions

- You have programmed before
- You understand objects & classes
- You can learn by osmosis

Code Organization

Packages, Classes, and Methods

Packages

- ...are used for namespacing
- ...correspond to folders

```
package com.pillartechnology.stuff;
```

Classes

- ...are units of code
- ...are contained in packages
- ...correspond to files

```java
package com.pillartechnology.stuff;

public class Stuff {
}
```

Methods

- ...do things
- ...are contained in classes

```
package com.pillartechnology.stuff;

public class Stuff {

  public void doStuff() {
    System.out.println("I do stuff");
  }

}
```

Types

- Types store stuff
- Many flavors
  - int & long
  - float & double
  - byte & char
  - boolean
- Classes are Types
  - String
  - Stuff

```
package com.pillartechnology.stuff;

public class Stuff {

  public void doStuff() {
    int dontPanic = 42;
    String message = "I do stuff";
    System.out.println(message);
  }

}
```

Methods: Return & Arguments

- Methods return things
- Methods take arguments

```
package com.pillartechnology.stuff;

public class Stuff {

  public int doStuff(String message) {
    int dontPanic = 42;
    System.out.println(message);
    return dontPanic;
  }

}
```

Fields

- ...store stuff for a class

```
package com.pillartechnology.stuff;

private int dontPanic = 42;
private String theMessage;

public class Stuff {

  public int doStuff(String message) {
    theMessage = message;
    System.out.println(message);
    return dontPanic;
  }

}
```

Classes: Creating & Using

- Classes are created with the new operator
- Methods on classes can be called

```
package com.pillartechnology.stuff;

public class Stuff {

  public int doStuff(String message) {
     ...
  }

}

...

Stuff stuff = new Stuff();
stuff.doStuff();  // returns 42
```

Classes: Constructors

- ...setup classes
- ...are invoked with the new operator

```
...

public class Stuff {

  public Stuff() {
    theMessage = "I do stuff";
  }

  public Stuff(int number, String msg)
{
    dontPanic = number;
    thisMessage = msg;
  }
  ...
}

Stuff stuff = new Stuff();
Stuff moreStuff = new Stuff(23,
"Boo!");
```

Imports

- ...make other classes available to the current class
- ...are not needed if classes are in the same package

```
package com.pillartechnology.stuff;

import com.pillartechnology.thing.
Thing;

public class Stuff {

  public int doStuff(String message) {
    Thing thing = new Thing();
    thing.doThings(message);
    return thing.getThings();
  }

}
```

Scoping

- ...hides methods and fields from outside tampering
  - public is public
  - private is private

```
public class Stuff {

  public String theMessage;
  private int theNumber;

  public void doStuff(String message) {
    doPrivateStuff(42);
  }

  private void doPrivateStuff(int i) {
  }
}

Stuff stuff = new Stuff();
stuff.theMessage;          // succeeds
stuff.theNumber;           // fails
stuff.doStuff();           // succeeds
stuff.doPrivateStuff(23);  // fails
```

Control Structures

if, else, for, while, do

If...Else

- Conditionally Runs Code
- Uses Operators to evaluate true
  - ==
  - !=
  - <
  - >
  - <=
  - >=
  - !

```
if (i == 42) {
  doStuff();
}

if (i != 42) {
  doStuff();
} else {
  doOtherStuff();
}

if (i < 12) {
  doStuff();
} else if (i < 24) {
  doOtherStuff();
} else {
  doThings();
}
```

Loops

- Loops until conditions met

```
int i;

i = 0;
while (i < 5) {
  doStuff();
  i++;
}

i = 0;
do {
  doStuff();
  i++;
} while (i < 5);

for (i = 0; i < 5; i++) {
  doStuff();
}
```

Arrays

Arrays

- ...are bounded sets of items of a particular type
- ...are zero-based
- ...work nicely with for loops

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

nums[0] = 1;
nums[1] = 2;

for (int i = 0; i < nums.length; i++) {
  nums[i] == i * 10;
}
```