Java Training

Day 5: Constructors, Variable Scope & Member Variables

Code examples are all single java file applications you are encouraged to copy-paste-compile-run and experiment with

Download today’s slides:
go/java+espresso-training/day5

Fun with Larry and Bob, Part #1 (a single Bob instance)

```
public class FunWithLB {

        public static void main(String[] args){
                System.out.println("Makin' Bobs");
                Bob bob1 = new Bob();
                System.out.println(bob1);
        }
}

class Bob {}   // No much here, just an empty Object subclass
```

```
> javac FunWithLB.java
> java FunWithLB

Makin' Bobs
Bob@43556938
```

Reference location for `bob1` since `Bob` does not overload `toString()`

Fun with Larry and Bob, Part #2 (two Bobs)

```
public class FunWithLB {

        public static void main(String[] args){
                System.out.println("Makin' Bobs");
                Bob bob1 = new Bob();
                System.out.println(bob1);

                Bob bob2 = new Bob();
                System.out.println(bob2);
        }
}

class Bob {}
```

```
> javac FunWithLB.java
> java FunWithLB

Makin' Bobs
Bob@43556938
Bob@3d04a311
```

Separate reference ids for `bob1` and `bob2`

These are what get compared when you use "==" with objects

Fun with Larry and Bob, Part #3 (self-aware Bob)

```
public class FunWithLB {

        public static void main(String[] args){
                System.out.println("Makin' Bobs");
                Bob bob1 = new Bob();
                System.out.println("bob1 says he's: " +
bob1.whoAmI());
        }
}

class Bob {
        String whoAmI(){
                return "Bob";
        }
}
```

```
> javac FunWithLB.java
> java FunWithLB

Makin' Bobs
bob1 says he's: Bob
```

Fun with Larry and Bob, Part #4 (Bob's son Larry)

```
public class FunWithLB {
        public static void main(String[] args){
                Larry larry1 = new Larry();
                System.out.println("larry1 says he's: " + larry1.whoAmI());
        }
}

class Bob {
        String whoAmI(){ return "Bob"; }
}

class Larry extends Bob {}
```

```
> javac FunWithLB.java
> java FunWithLB

larry1 says he's: Bob
```

What's going on with the Larry?

- How can send a whoAmI() to him
- Why does he say he's "Bob"?
- Can Larry be fixed?

Fun with Larry and Bob, Part #5
(Larry learn who he is)

```
public class FunWithLB {
        public static void main(String[] args){
                Larry larry1 = new Larry();
                System.out.println("larry1 says he's: " + larry1.whoAmI());
        }
}

class Bob {
        String whoAmI(){ return "Bob"; }
}

class Larry extends Bob {
        String whoAmI(){ return "Larry"; }
}
```

```
> javac FunWithLB.java
> java FunWithLB

larry1 says he's: Larry
```

Fun with Larry and Bob, Part #6 (Larry knows his roots)

```
public class FunWithLB {
        public static void main(String[] args){
                Larry larry1 = new Larry();
                System.out.println("larry1 says he's: " +
larry1.whoAmI());
        }
}

class Bob {
        String whoAmI(){ return "Bob"; }
}

class Larry extends Bob {
        String whoAmI(){ return "Larry son of " + super.whoAmI(); }
}
```

```
> javac FunWithLB.java
> java FunWithLB

larry1 says he's: Larry son of Bob
```

Scope and Member Variables, part #1

In addition to *methods*, classes can have **Member Variables** which can give instances **state**

```
public class Scoping {
        public static void main(String[] args){
                Bob bob1 = new Bob();
                bob1.age = 34;
                System.out.println("bob1 is " + bob1.age);
        }
}

class Bob {
        int age;
}
```

As with methods, *member variables* are accessed using **dot-notation**

```
> javac Scoping.java
> java Scoping

bob1 is 34
```

Scope and Member Variables, part #2

Instances can then have *unique* **state**

```
public class Scoping {
        public static void main(String[] args){
                Bob bob1 = new Bob();
                Bob bob2 = new Bob();
                bob1.age = 34;
                bob2.age = 18;
                System.out.println("bob1 is " + bob1.age);
                System.out.println("bob2 is " + bob2.age);
        }
}

class Bob {
        int age;
}
```

```
> javac Scoping.java
> java Scoping

bob1 is 34
bob2 is 18
```

How would you determine if two Bob instances were *equal*?

A bit more about Equality

```
public class Scoping {
        public static void main(String[] args){
                Bob bob1 = new Bob();
                Bob bob2 = new Bob();
                bob1.age = 4;
                bob2.age = 4;
                System.out.println(bob1.equals(bob2));
        }
}

class Bob {
        int age;
        public boolean equals(Object object){
                return (this.age == ((Bob)object).age);
        }
}
```

There is brittleness here

```
> javac Scoping.java
> java Scoping

true
```

Without the equals implementation, you would get *false*

A bit more about Equality

```
class Bob {
        int age;
        public boolean equals(Object object){

                return (object instanceof Bob &&
                        this.age == ((Bob)object).age);
        }
}
```

This won’t crash if you call it with a non-`Bob` object

Final point about object equality (Introspection)

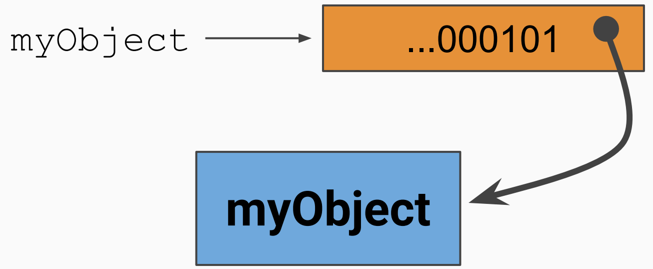
Compare objects using equals()

```
someObject.equals(anotherObject)
```

Calling `equals()` is asking the object to define equality for itself

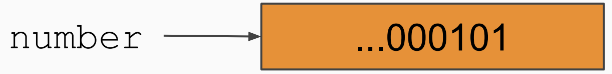
Object Types
(the object defines “equals”)

```
SomeClass myObject = new SomeClass();
```



Primitive Types
(equality is obvious, use “==”)

```
int number = 5;
```



Scope and Member Variables, part #3

Member variables are available directly to instance methods

```
public class Scoping {
        public static void main(String[] args){
                Bob bob1 = new Bob();
                bob1.age = 34;
                bob1.doSomething();
        }
}

class Bob {
        int age;
        void doSomething() {
                System.out.println("my age is " + age);
        }
}
```

```
> javac Scoping.java
> java Scoping

my age is 34
```

Scope and Member Variables, part #4

Member variables may be *shadowed* by local variables

```
public class Scoping {
        public static void main(String[] args){
                Bob bob1 = new Bob();
                bob1.age = 34;
                bob1.doSomething();
        }
}

class Bob {
        int age;
        void doSomething() {
                int age = 5;
                System.out.println("my age is " + this.age);
        }
}
```

```
> javac Scoping.java
> java Scoping

my age is 34
```

Use the `this` reference to explicitly reference a member variable

Without `this`, it will output `5`

Scope and Member Variables, part #5

Scopes are defined by Curly Brackets...
...but indentation allows you to keeps your sanity

```
{scope 1 {scope 2 {scope 3}} {scope 4 {scope 5 {scope 6}}}}

           |---- scope 3 --|           |------ scope 6 -------|

                      |----------- scope 5---------------------|

     |----------- scope 2-----------------|  |-------------------------------scope 4 -----------------------|

 |-------------------------------------------------------- scope 1 ----------------------------------------------------------------|
```

Constructors, the return of Bob

It’s convenient to set the state of an instance when it’s created

```
public class Constructor {
        public static void main(String[] args){
                Bob bob1 = new Bob(64);
                System.out.println(bob1.age);
        }
}

class Bob {
        int age;
        public Bob(int age){
                this.age = age;
        }
}
```

A **constructor** can have as many **arguments** as you like and you may have as many constructors as you have **unique argument lists**

```
> javac Constructor.java
> java Constructor

64
```

Bob with Constructor and toString() override

```
public class Constructor {
        public static void main(String[] args){
                Bob bob1 = new Bob(64);
                System.out.println(bob1);
        }
}

class Bob {
        int age;
        public String toString(){
                return "Bob: "+age+" yrs";
        }
        public Bob(int age){
                this.age = age;
        }
}
```

```
> javac Constructor.java
> java Constructor

Bob: 64 yrs
```