Override, scope & packages

Ch 5.5

DEVELOPER

Y U NO PROTECT MY DATA?
Highlights

- Overriding functions

```java
class Parent {
    public void foo() {
        System.out.println("foo");
    }
}

class Child extends Parent {
    @Override
    public void foo() {
        System.out.println("bar");
    }
}
```

- protected / package scope

```java
protected int x;
int y;
```
Review: Inheritance
Class review

4 Major Principles for OOP
Inheritance

```java
public class Burger

public class BaconBurger extends Burger
```
Override

Child classes can **override** methods from their parent's class.

This means a child can redefine a method to work in a completely different way than the parent's method.

Can use **super** to get the parent's version of an overridden method.
Override

To override, you simply have the same name of a function as your parent (and no overload)

```java
public class Parent {
    public void foo() {
        System.out.println("foo");
    }
}
```

```java
public class Child extends Parent{
    @Override
    public void foo() {
        System.out.println("bar");
    }
}
```

Optional (but recommended)

(See: OverrideSandworm.java)
You should always add `@Override` before every method you override (as a sanity check)

When overriding, you can change modifiers and return types under these restrictions:

1. The modifier must be more “open” (e.g. private -> public)
2. The return type is a descendant (child) of the original return type (e.g. Parent -> Child)

(See: OverrideChild.java and OverrideParent.java)
Override vs Overload

When you **override**, you redefine an existing method (name), i.e. you replace a method (same name, number of arguments and types).

When you **overload**, you add a new way of using a method (name), i.e. add alternative (same name, but different arguments).
We have used `final` to declare constants (objects who's value cannot change)

You can use the `final` modifier with methods and classes as well:

- `final` classes cannot be the parent for any class
- `final` methods cannot be overridden

(See: Final.java, FinalClass.java and FinalMethod.java)
Scope modifiers

**private** - Only usable inside original class

```java
private int x;
```

(blank) - Usable by any class in the package

```java
int x;
```

**protected** - Usable by any descendant (e.g. child) or any class in the package

```java
protected int x;
```

**public** - Usable by any class

```java
public int x;
```

(When overriding, can change down this list)
Scope modifiers

<table>
<thead>
<tr>
<th>Var</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.x</td>
<td>Any class</td>
</tr>
<tr>
<td>A.y</td>
<td>Only A</td>
</tr>
<tr>
<td>B.y</td>
<td>Any class</td>
</tr>
<tr>
<td>B.z</td>
<td>B, C and any in package</td>
</tr>
<tr>
<td>C.w</td>
<td>Any in package</td>
</tr>
<tr>
<td>D.z</td>
<td>Any in package</td>
</tr>
<tr>
<td>E.x</td>
<td>Any class</td>
</tr>
</tbody>
</table>

(See: A, B, C, D.java)
Scope modifiers

Java Scope

Package A (or default package)

Class1

(default)

private

public

protected

Package B

Class3
Sub Class of Class1

Class2

Class4
Packages? Class? Inheritance?

Classes are a way of grouping multiple objects to the same abstract idea (e.g. Person has a name and height)

Inheritance is used to show relationships between classes and allows code to be reused

Packages group classes that do similar things (e.g. car.fluid and car.parts)
Why should I care about scope?

If you are writing a simple program that only you, yourself, will use... then you do not.

But imagine if all programs gave full access to everything...