

Review

Let's look at the following code:

```
class Person {
    String name;
    PersonList kids;
    Person( String name, PersonList kids ) {
        ...
    }
    int numKids() {
        return this.kids.num();
    }
}
...
class LargerPL extends PersonList {
    Person first;
    PersonList rest;
    ...
    int num() {
        return 1 + first.numKids() + rest.num();
    }
}
```

What's missing?

```
class Person {
    String name;
    PersonList kids;
    Person( String name, PersonList kids ) {
        ...
    }
    boolean sameName(String n) ...
    int numKidsWithName(String n) {
        return this.kids.numWithName(n);
    }
}
...
class LargerPL extends PL {
    Person first;
    PersonList rest;
    ...
    int numWithName(String name) {
        if ( first.sameName(name) )
            return 1 + rest.numWithName(name);
        else
            return rest.numWithName(name);
    }
}
```

Cyclic call

```
class Person {
    String name;
    PersonList kids;
    Person( String name, PersonList kids ) {
        ...
    }
    boolean sameName(String n) ...
    int numKidsWithName(String n) {
        return this.kids.numWithName(n);
    }
}
...
class LargerPL extends PL {
    Person first;
    PersonList rest;
    ...
    int numWithName(String name) {
        if ( first.sameName(name) )
            return 1 + first.numKidsWithName(name) + rest.numWithName(name);
        else
            return first.numKidsWithName(name) + rest.numWithName(name);
    }
}
```

A familiar example

- Our bank contracted a web development company to build a e-commerce site for us.
- Our task was to design the BankAccount representation for the site and internal accounting

BankAccount
int idNumber
double balance
void deposit(double amt)
boolean withdraw(int id, double amt)
void addInterest()
• String viewBalance(int id)

Time passes

- Our customers happily access their accounts on-line
- We pay the web development company, and all go are merry ways

Modifying our design

- The head honcho wants the representation changed
- Using a double allows people to get fraction of cents interest
 - The Boss wants it stopped
- Our task, modify the representation
 - So that the boss is happy
 - Without affecting the website -- method headers must remain the same

BankAccount
int idNumber
int numCents
void deposit(double amt)
boolean withdraw(int it, double amt)
void addInterest()
◦ String viewBalance(int id)

Whoops

- Little did we know ...
- This was our web development partner's code:
 - class `WebApp` {
 ...
 HTMLBox showAvailBalance(BankAccount b) {
 ... b.balance ...
 }
}
- They messed up.
- But **OUR** business is hurt when the website goes down
 - Could we have prevented this?

public/private

- public
 - classes, fields, methods, constructors
 - Everyone has full access (including mutation)
- private
 - fields, methods, constructors
 - No one outside of the declaring class may access them
- protected
 - fields, methods, constructors
 - Subclasses may access
- No modifier means default access -- we'll discuss in a few slides

Let's get in the Way back machine

- We don't want our web development team relying on the balance field
- Or doing something inappropriate with accounts, balances, or gaining interest

BankAccount
private int idNumber private double balance
public void deposit(double amt) public boolean withdraw(int id, double amt) protected void addInterest() public String viewBalance(int id)

Let's get in the Way back machine

- We don't want our web development team relying on the balance field
- Or doing something inappropriate with accounts, balances, or gaining interest

BankAccount
- int idNumber - double balance
+ void deposit(double amt) + boolean withdraw(int id, double amt) o void addInterest() + String viewBalance(int id)

- Now our partner's mistake can't happen

Seeing the Java code

```
public class BankAccount {  
    private int idNumber;  
    private double balance;  
    BankAccount( int idNumber, double balance ) { ... }  
  
    public void deposit( double amt ) { ... }  
    protected addInterest() { ... }  
}
```

Deciding without a Way Back machine

Guidelines, not rules

- Methods that provide core functionality
 - public
- Methods that support core functionality, and have no special side-effects
 - default access
- Methods that support core functionality but have some side-effects
 - protected
- Methods that need fine-grained control
 - private

Deciding without a Way Back machine

Guidelines, not rules

- Fields that hold highly useful constants
 - public
- Fields that are highly useful to others, in designs that are unlikely to change
 - public, default
- Fields whose representations are likely to change
 - private
- Fields that contain secret information
 - private

Revisiting another old friend (expanded)

How should we modify our design

```
Fish
int weight
int tagNo
Food favorite
Fish feedFish( Food served )
boolean isFish( int id )
boolean weighsMore( int than )
void changeFavorite( Food newFavorite )
```

You changed my constant!

- Remember the cautions about mutation?
double pi = 3.14159;
.....
pi = 3;
- Can access controls prevent this problem?
 - Yes, private variables cannot be mutated by outsiders

But

pi is potentially a very useful constant to provide

Java provides a means to prevent unwanted mutations

final

- A final field must be set immediately or in the constructor
- A final field can never be mutated again

final double pi = 3.14159;
- Methods can be final as well
 - Final methods cannot be overridden by a subclass

Libraries

- public/private/final -- design decisions
 - Affect how other classes in the program can interact with yours
 - Affects how other programmers will use libraries you provide
- Important to understand in using libraries
 - What functionality is available
 - How should my program interact with this library
 - What responsibilities do I have

Designing a library

RGBColor
int maxIntensity
int minIntensity
int red
int green
int blue
RGBColor mixColor(int r, int g, int b)
RGBColor black()
RGBColor red()

Using a Java library

- When inheriting, methods must have at least the same access as in the parent
- Primarily, public methods provide functionality, fields provide constants
- Many Java libraries available
 - We need a few more pieces first

Grouping classes into libraries

- One class normally does not contain all provided functionality
 - One file usually does contain only one class
- A library is a group of related (often inter-working) classes
 - In Java, called a package
- All files are members of a package
 - package `myCompany.myProduct`;
 - Appears at the very top of a file
- Files without a specification form a default package

default access

- Remember that fields, methods, etc, without a modifier had default access
- package members have default access
- package members can access protected items as well
- private -> default -> protected -> public

Constants

- Consider our `int maxIntensity`; field
 - A constant value
 - The value is not tied to any one instance of `RGBColor`
 - Could be used in creating instances of `RGBColor`

Does it make sense for this to be a member of `RGBColor`?

static

- Another modifier of fields and methods
- Connected to the class
- Not connected to the instance
 - You don't need an object to access the value

An example

```
public class RGBColor {
    public static final int minIntensity = 0;
    public static final int maxIntensity = 255;
    int red, green, blue;
    public RGBColor( int red, int g, int b ) { ... }
}
public class ClownNose {
    RGBColor c = new RGBColor( RGBColor.maxIntensity,
                               RGBColor.minIntensity,
                               RGBColor.minIntensity);
}
```

static

- Terminology
 - static int numCars;-- static
 - int numDoors;-- instance
- static entities cannot directly access instance fields/methods
 - Incorrect

```
public class ClownNose {
    RGBColor c = new RGBColor( RGBColor.maxIntensity,
                               RGBColor.minIntensity,
                               RGBColor.minIntensity);
    static int redLevel = c.red;
}
```

static

- Terminology
 - static int numCars;-- static
 - int numDoors;-- instance
- static entities cannot directly access instance fields/methods
 - Correct

```
public class ClownNose {
    RGBColor c = new RGBColor( RGBColor.maxIntensity,
                               RGBColor.minIntensity,
                               RGBColor.minIntensity);
    static int redLevel = new ClownNose().c.red;
}
```

Using libraries : take 2

- Libraries provide a lot of handy data structures/functionality
- Java implementations come with a large set of libraries
 - <http://java.sun.com/j2se/1.4.2/docs/api/index.html>

java.lang.System & java.io.PrintStream

- These libraries provide support for printing
 - Will be used in your projects
- java.lang.System
 - Automatically imported into all programs -- all of java.lang is
 - Contains support for interacting with the outside world
 - One notable field : public static PrintStream out
- java.io.PrintStream
 - Also does not need to be imported to print
 - Contains methods to display content to the screen
 - public void println(XXXX)
 - XXXX can be any value in Java

Printing

```
System.out.println( "Hello World!");
System.out.println( new Fish(4) );
System.out.println( 3 );
```

Writing a program in full Java

- Much like what we have been doing
- But requires a special method in order to run
 - `public static void main(String[] args) { ... }`
- The body of this method is what is run by java
- `args --` contains any command line arguments present when the program is called

A sample program

Old style

```
class Fish {
    int weight;
    Fish( int weight ) {
        this.weight = weight;
    }
    Fish grow( int byAmt ) {
        return new Fish( this.weight + byAmt );
    }
    boolean tooBig() {
        return this.weight > 10;
    }
}
```

```
Fish f = new Fish(5);
f
f.tooBig()
f.grow(10)
f.grow(5).tooBig()
```

A sample program

New style

```
public class Fish {
    private int weight;
    public Fish( int weight ) {
        this.weight = weight;
    }
    public Fish grow( int byAmt ) {
        return new Fish( this.weight + byAmt );
    }
    public boolean tooBig() {
        return this.weight > 10;
    }
    public String toString() {
        return "Fish("+this.weight+")";
    }
    public static void main( String[] args ) {
        Fish f = new Fish(5);
        System.out.println(f);
        System.out.println(f.tooBig());
        System.out.println(f.grow(10));
        System.out.println(f.grow(5).tooBig());
    }
}
```

A few more...

Let's look at a few more examples

