Topic 13: Classes

- Why Classes?
- private vs. public
- Gettors and Settors
- this
- static (Reprise)
- Constructors
- toString()
Why Classes?

• Classes are a way of organizing code into **logical units**
• Code can interact, but doesn't conflict
• Code connects through **interfaces**
Why Classes?

We use interfaces all the time in the real world:

- We know what these pedals mean.
- We know how to use them.
- But we don't need to know how they work internally.
Why Classes?

What's the proper way to drive a car?

Well-defined interfaces make it easier to complete complex tasks.
Why Classes?

- A well designed class will distinguish between:
  - Public interfaces (for external use)
  - Private implementation (hidden)

<table>
<thead>
<tr>
<th>Public:</th>
<th>Private:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerator</td>
<td>Fuel injectors</td>
</tr>
<tr>
<td>Brake</td>
<td>Antilock system</td>
</tr>
<tr>
<td>Speedometer</td>
<td>Transmission</td>
</tr>
<tr>
<td>Fuel filler pipe</td>
<td>Fuel tank</td>
</tr>
</tbody>
</table>
Why Classes?

• Public interfaces do various things:
  – Change the state
  – Query the state
  – Perform an action

• Private implementation can do anything:
  – But the outside world doesn't care about it!
Topic 13: Classes

- Why Classes?
- `private` vs. `public`
- Gettors and Settors
- `this`
- `static` (Reprise)
- Constructors
- `toString()`
private vs. public

- **public**:  
  - Anybody can use it

- **private**:  
  - This class only

- **protected**:  
  - Something in-between (not discussed here)
private vs. public

- Java forbids access to private methods and data from outside the class

```java
public class Foo {
    private int x;
}

public class Bar {
    public void method(Foo f) {
        f.x = 10; // error!
    }
}
```
**private vs. public**

- Methods **inside** the class have free access to **private methods and data**

```java
public class Foo {
    private int x;

    public void incrementX() {
        x++;
    }
}
```

This is a very common strategy.

A **public method** does things which modify **private data**.
private vs. public

- Private helper methods are often handy.

```java
public class Foo {
    private int x;
    public void incrementX() {
        addToX(1);
    }
    private void addToX(int i) {
        x += i;
    }
}
```

`addToX()` is trivial, but my slides are small!

Helper methods are often a good way to handle small sub-tasks.
Examples of Encapsulation

- **String**
  - **public:** `charAt()`, `substring()`, `length()`
  - **private:** we don't know

- **Scanner**
  - **public:** `hasNext()`, `next()`, `nextInt()`
  - **private:** we don't know
Why Not Make It All `public`?

- `public` is fine for little projects
  - Example: Things we write in this class
- Bad idea for huge projects
  - Example: `String` class

- Once a field is `public`, it's hard to change
  - People write code that directly access public fields
  - Changing the fields breaks other people's code
Why Not Make It All public?

- private fields can change at any time!
  - Remove them
  - Rename them
  - Change types
  - Change semantics

- You'll never break somebody else's code
Topic 13: Classes

- Why Classes?
- `private` vs. `public`
- **Gettors and Settors**
- `this`
- `static` *(Reprise)*
- **Constructors**
- `toString()`
Good Practice

- Most data fields should be private
- Use getters and setters to access data
  - Getter: gets the value of a private variable
  - Setter: sets the value of a private variable
Gettors and Settors

Why gettors and settors?

• Later, you might change the implementation
• Can do error checking
• Can set other fields that might be related

• Can keep a field secret (don't define a getter)
• Can ban changing a field (don't define a setter)
public class ReallySimple
{
    private int val;

    public void setVal(int newVal)
    {
        val = newVal;
    }

    public int getVal() { return val; }
}
public class ReallySimple
{
    private int val;

    public void setVal(int newVal)
    {
        val = newVal;
    }

    public int getVal() { return val; }
}

The data element is private.
You cannot access it from outside this class.
Gettors and Settors

public class ReallySimple {

    private int val;

    public void setVal(int newVal) {
        val = newVal;
    }

    public int getVal() { return val; }

}

The settor is public.

It has a parameter, and returns void.
public class ReallySimple
{
    private int val;

    public void setVal(int newVal)
    {
        val = newVal;
    }

    public int getVal() { return val; }
}
Topic 13: Classes

- Why Classes?
- `private` vs. `public`
- Gettors and Settors
- `this`
- `static` (Reprise)
- Constructors
- `toString()`
• An instance method has access to the instance variables (and methods)

• But how?
  - To access the fields of an object, we need an object reference.
  - How does the method know which object it is accessing?
public class ReallySimple
{
    private int val;

    public void setVal(int newVal)
    {
        val = newVal;
    }

    public int getVal() { return val; }
}

How do these methods find out where val is?
public class ReallySimple {

    private int val;

    public void setVal(ReallySimple this, int newVal) {
        this.val = newVal;
    }

    public int getVal(ReallySimple this) {
        return this.val;
    }
}

Every instance method has a secret parameter named *this*. It tells the method which object to access.
public class ReallySimple {

    private int val;

    public void setVal(ReallySimple this, int newVal) {
        this.val = newVal;
    }

    public int getVal(ReallySimple this) {
        return this.val;
    }
}
Using **this**

- **It's never legal** to declare a parameter named `this`.
  - Compile error

- But it's OK to **read** it in your code!
  - Only inside instance methods
public class UsingThis1
{
    private int val;
    public int getVal() { return val; }
    public void setVal(int val)
    {
        this.val = val;
    }
}

There are two variables named val. setVal() can access both.

Which should it use?
Using this

```java
public class UsingThis1 {
    private int val;

    public int getVal() { return val; }

    public void setVal(int val) {
        this.val = val;
    }
}
```

When there are duplicate names, the one without this refers to the parameter (or local variable).
Using this

```java
public class UsingThis1 {
    private int val;
    public int getVal() { return val; }
    public void setVal(int val) {
        this.val = val;
    }
}
```

`this.val` always refers to an instance field, no matter what.
Is this Ever Required?

- We've seen `this` used to resolve name ambiguity
- But we could have avoided that with a better parameter name

- Is the keyword `this` every really `necessary`?
  Yes!
Sometimes, we want to store a reference to the current object into a variable
  - Very useful for data structures!

this is a reference, which we can store (or use for comparisons), a lot like null.
  - It's never legal to modify this
Topic 13: Classes

- Why Classes?
- `private` vs. `public`
- Gettors and Settors
- `this`
- `static` (Reprise)
- Constructors
- `toString()`
static (Reprise)

- A static method is not associated with any particular object.

- What about this?
  - A static method doesn't have a this parameter!
public class ReallySimple_StaticBug
{
    private int val;

    public static void setVal(int newVal)
    {
        val = newVal;  // error!
    }

    public int getVal() { return val; }
}

setVal() is static. Thus, it has no this reference – and it does not know where to find val.
Calling Methods

- To call an instance method:
  
  ```java
  obj.instanceMethod(1,2);
  ```

- To call a static method:
  
  ```java
  MyClass.staticMethod(3,4);
  ```

In Class:
How many parameters are passed to each method?
Calling Methods

• To call an instance method:
  \texttt{obj.instanceMethod(1,2);}

  This method has 3 parameters (including \texttt{this})

• To call a static method:
  \texttt{MyClass.staticMethod(3,4);}

  This method has 2 parameters
static Variables

- A `static` variable is one which is shared across the whole class. There is only one copy.

- Access it just like a `static` method:
  ```java
  MyClass.instanceVariable = 3;
  ```

- `static` variables can be `public` or `private`.
Topic 13: Classes

- Why Classes?
- private vs. public
- Gettors and Settors
- this
- static (Reprise)
- Constructors
- toString()
What is a Constructor?

• Method called to initialize an object
  - Called by new

• Like a normal method, except:
  - Must always have same name as class
  - Must not have any return type (not even void)
  - Can never be called directly!
public class Rational1
{
    private int num, den;

    public int getNum() { return num; }
    public int getDen() { return den; }

    public Rational1(int num, int den)
    {
        this.num = num;
        this.den = den;
    }

    public String toString() { ... }
}
public class ConstructorMain
{
    public static void main(String[] args)
    {
        Rational1 obj1 = new Rational1(3,5);
        Rational1 obj2 = new Rational1(0,10);
        Rational1 obj3 = new Rational1(10,0);

        System.out.println("obj1: "+obj1);
        System.out.println("obj2: "+obj2);
        System.out.println("obj3: "+obj3);
    }
}

Create a Rational1 object; pass 3,5 to its constructor.
public class ConstructorMain {
    public static void main(String[] args) {
        Rational1 obj1 = new Rational1(3,5);
        Rational1 obj2 = new Rational1(0,10);
        Rational1 obj3 = new Rational1(10,0);

        System.out.println("obj1: " + obj1);
        System.out.println("obj2: " + obj2);
        System.out.println("obj3: " + obj3);
    }
}
Constructors Initialize an Object

Let's examine what the stack looks like when we create an object that has a constructor.

We start with `main()`, which is about to allocate an object with `new`.

```c++
main():
obj1 =
obj2 =
Obj3 =
```
Constructors Initialize an Object

Stack

```
main():
$obj1 =
$obj2 =
.Obj3 =
```

Rational1
num = 0
den = 0

`new` is code that is provided by Java.

It is almost certainly implemented as a secret method (which we never see directly).
Constructors Initialize an Object

Stack

Rational1():
  this =
  num = 3
  den = 5

<new>

main():
  obj1 =
  obj2 =
  Obj3 =

new calls the constructor.

It passes a this parameter, plus any parameters that we gave to new.
Constructors Initialize an Object

The constructor is like any other method.

I uses the this reference to update the fields.
Constructors Initialize an Object

When we return to `main()`, it gets a reference to the newly allocated object.

```
main():
obj1 =
obj2 =
Obj3 =
```

```
Rational1
num = 3
den = 5
```
Constructors Can Throw Exceptions

```java
public Rational1(int num, int den)
{
    if (den == 0)
        throw new IllegalArgumentException();

    this.num = num;
    this.den = den;
}
```

Just like any other method, a constructor can throw an exception.
Constructors that Throw Exceptions

- Where have you seen a constructor that throws exceptions?

`Scanner(File) throws FileNotFoundException`
Constructors Can Do Logic

```java
public Rational1(int num, int den)
{
    if (den == 0)
        throw new IllegalArgumentException();

    if (num == 0)
        den = 1;

    this.num = num;
    this.den = den;
}
```

A constructor is like a gigantic settor.

It can include any legal Java code – if(), loops, even creating other objects!
Constructor Can Allocate Objects

```java
public Voter(String name) {
    this.name = name;
    this.preferences = new String[5];
}
```

If you have an array instance variable, it's often a good idea to **initialize it** in the constructor.

This helps avoid `NullPointerException`. 
Topic 13: Classes

- Why Classes?
- `private` vs. `public`
- Gettors and Settors
- `this`
- `static` (Reprise)
- Constructors
- `toString()`
```java
toString()

- By default, when you concatenate a String and an object, you get a nasty mess:

```java
MyClass obj = new MyClass();
obj.x = 10;
System.out.println("obj="+obj);
```

**Output:**

```
obj=MyClass@7c9a2f1a
```
You can override this behavior, so that you get nice printouts:

```java
MyClass obj = new MyClass();
obj.x = 10;
System.out.println("obj="+obj);
```

**Output:**

```
obj={x=10 y=0}
```
toString()

• How? With `toString()`!

• Add the following method to any class:

  ```java
  public String toString() { ... }
  ```

• Java will automatically find it, and use it for concatenation.
public class MyClass
{
    public int x;
    public int y;

    public String toString()
    {
        return "{x="+x+" y="+y+"}";
    }
}

Put any code you want into toString().

What you return from the method will be used as the “String version” of your object (for concatenation).
How does `toString()` work?

How does Java find your `toString()` method?

What happens when you don't define `toString()`?

That topic requires that you understand inheritance ... which you will learn later.
Topic 13: Classes

- Why Classes?
- `private` vs. `public`
- Gettors and Settors
- `this`
- `static` (Reprise)
- Constructors
- `toString()`

Summary