Objectives

- Polymorphism
  - Dispatch
- Inheritance
  - Final methods, fields
- Eclipse
- Javadocs

Review

- How does Java pass parameters?
- How does a class refer to its parent class?
- What does a class inherit from its parent class?
  - What is *not* inherited?
- What are the access modes, ordered from least restrictive to most restrictive?

Code Review

- What were the datatypes of your `Birthday` class's instance variables?
- Why do I like this method?

```
public void changeDay(int dayUpdate){
  if ( (dayUpdate < 32) && (dayUpdate > 0) ) {
     day = dayUpdate;
  }
  else {
     System.out.println(dayUpdate + " is not a valid “
           + “day");
  }
}
```

Epitome of benefits of encapsulation

- How could the method be improved?

Code Review

```
public static void main(String[] args) {
   Birthday birthday = new Birthday (”Sept", 25);
   System.out.println(”My birthday is " +
      birthday.getMonth() + birthday.getDay() + ".”);
   …
}

public String getMonth() {
   return month + " ";
}
```

- Discuss this API and how it would be used

Assignment Review

```
public static void main(String[] args) {
   Birthday birthday = new Birthday (”Sept", 25);
   System.out.println(”My birthday is " +
      birthday.getMonth() + birthday.getDay() + ".”);
   …
}

public String getMonth() {
   return month + " ";
}
```

- `getMonth()` probably does not behave as user expects

```
if( birthday.getMonth().equals(“September” )) {
      // print Happy Birthday Month!
}
```


POLYMORPHISM & DISPATCH

Polymorphism

- You can use a child class object whenever the program expects an object of the parent class
- Object variables are *polymorphic*
- A `Chicken` object variable can refer to an object of class `Chicken, Hen, Rooster`, or any class that *inherits from* `Chicken`

Polymorphism

```
Chicken[] chickens = new Chicken[3];
chickens[0] = momma;
chickens[1] = foghorn;
chickens[2] = baby;
```

- We can guess the actual types
- *But compiler can't*

Polymorphism

```
Chicken[] chickens = new Chicken[3];
chickens[0] = momma;
chickens[1] = foghorn;
chickens[2] = baby;
```

- We know `chicken[1]` is probably a `Rooster`, but to *compiler*, it's a `Chicken` so `chicken[1].crow();` will not compile

Polymorphism

- When we refer to a `Rooster` object through a `Rooster` object variable, compiler sees it as a `Rooster` object
- If we refer to a `Rooster` object through a `Chicken` object variable, compiler sees it as a `Chicken` object.
- We cannot assign a parent class object to a derived class object variable
  - A `Rooster` is a `Chicken`, but a `Chicken` is not necessarily a `Rooster`

Polymorphism

- Which method do we call if we call `chicken[1].feed()`
  Rooster's or Chicken's?

Polymorphism

- Which method do we call if we call `chicken[1].feed()`
  Rooster's or Chicken's?
- In Java: Rooster's!
  - Object is a `Rooster`
  - The JVM figures out its class at runtime and runs the appropriate method
- **Dynamic dispatch**
  - At runtime, the object's class is determined
  - Then, the appropriate method for that class is dispatched

Dynamic vs. Static Dispatch

- Dynamic dispatch is a property of Java, not object-oriented programming in general
- Some OOP languages use **static dispatch** where the type of the object variable used to call the method determines which version gets run
- The primary difference is ***when* decision on which method to call is made**…
  - Static dispatch (C#) decides at compile time
  - Dynamic dispatch (Java, Python) decides at run time
- Dynamic dispatch is slow
  - In mid to late 90s, active research on how to decrease time

Feed the Chickens!

Recall:

```
Chicken[] chickens = new Chicken[3];
chickens[0] = momma;
chickens[1] = foghorn;
chickens[2] = baby;
```

```
for( Chicken c: chickens ) {
    c.feed();
}
```

How to read this code?

- Dynamic dispatch calls the appropriate method in each case, corresponding to the actual class of each object.
  - This is the power of polymorphism and dynamic dispatch!

What Will This Code Output?

```
class Parent {
    public Parent() {}

    public void method1() {
        System.out.println("Parent: method1");
    }

    public void method2()
        System.out.println
        method1();
    }
}

class Child extends Paren
    public Child() {}

    public void method1()
        System.out.println
    }
}
```

```
public class DynamicDispatchExample {
    public static void main(String[] args) {
        Parent p = new Parent();
        Child c = new Child();

        p.method1();
        System.out.println("");

        c.method1();
        System.out.println("");

        p.method2();
        System.out.println("");

        c.method2();
        System.out.println("");
    }
}
```

See handout

Inheritance Rules: Access Modifiers

Access modifiers in child classes
- Can make access to child class ***less*** restrictive but not more restrictive

- **Why?**
- Consider what would happen if a method in the parent class is `public` but the child class is `private`

Inheritance Rules: Access Modifiers

Access modifiers in child classes
- Can make access to child class ***less*** restrictive but not more restrictive

- If a `public` method could be overridden as a `protected` or `private` method, child objects would not be able to respond to the same method calls as parent objects → **breaks polymorphism**
- When a method is declared `public` in the parent, the method remains `public` for all that class's child classes
- Remembering the rule: compiler error to override a method with a more restricted access modifier

CASTING

Explicit Object Casting

- Just like we can cast variables:

```
double pi = 3.14;
int i_pi = (int) pi;
```

- We can cast objects.

```
Rooster foghorn = (Rooster) chickens[1];
```

  - Use casting to use an object in its full capacity after its actual type (the derived class) has been forgotten

Example: Explicit Object Casting

- `Rooster` object is referred to only using a `Chicken` object variable
  - `chickens[1]` is an object variable to a `Chicken` object
  - We cannot access any `Rooster`-specific fields or methods using this object variable
- Create new object variable to `Rooster` object
  - This variable allows us to reference the Rooster-specific fields and methods…

```
Rooster rooster = (Rooster) chickens[1];
```


Object Casting

- We can do explicit type casting because `chickens[1]` refers to an object that is actually a `Rooster` object
- For example, cannot do this with `chickens[0]` because it refers to a Hen (not `Rooster`) object

```
Rooster rooster = (Rooster) chickens[1];
      // OK; chickens[1] refers to a Rooster object
Rooster hen = (Rooster) chickens[0];
      // ERROR; chickens[1] refers to a Hen object
```

- Promising compiler that although `chickens[1]` is an object variable to a `Chicken` object, it really refers to a `Rooster` object,
- If this is not the case, generates an exception
  - More about exceptions later

`instanceof` Operator

- Use `instanceof` operator to make sure such a cast will succeed

```
if (chickens[1] instanceof Rooster) {
     rooster = (Rooster)chickens[1];
}
```

- Operator returns a `boolean`
  - true iff `chickens[1]` refers to an object of type `Rooster`
  - false otherwise

Update `equals` in `Chicken.java`

Summary of Inheritance

- Place common operations & fields in parent class
  - Remove repetitive code by modeling the "is-a" hierarchy
  - Move "common denominator" code up the inheritance chain
- Don't use inheritance unless *all* inherited methods make sense
- Use polymorphism

JAVADOCS

Javadocs

- Special comments, which are used to generate HTML documentation
- Syntax:

```
/**
 * Comment
 */
```

- Put before a class, a method, or a field to describe the respective class/method/field

Javadoc

- Can contain HTML syntax in description
- Example block comments to help describe your code

```
@param <paramname> <description>
@return <description>   (include special cases)
@author <author's name>
```

  - More that we'll get to later

Examples

```
/**
 * A simple Java class that models a Chicken.  The
 * state of the chicken is its name, height, and weight
 *
 * @author Sara Sprenkle
 */
```

```
/**
 * @return the height of the chicken, in centimeters
 */
```

```
/**
 * @param n the String representing the name of the
chicken
 */
```


ECLIPSE

http://www.eclipse.org/

- Open source integrated development environment (IDE) for Java
- Has market share for Java IDEs
- Described as "an open extensible IDE for anything and nothing in particular"
- Provides a robust Java development environment
- Incorporates popular software development tools like JUnit and CVS
  - More on those later this semester
- Plugins allow extensibility

Project/Code Organization

- `workspace` directory contains all projects
  - Located in your home directory, unless you specified otherwise
- Use **projects** to organize your code
- Within a project
  - `src/` directory contains `.java` files
  - `bin/` directory contains `.class` files

Java Made Easier

- Creating class's basic functionality
  - See `Source` and `Refactor` menus
- Gives you a list of methods for an object
  - After you type `object.`
  - Then shows parameters for methods
- Automatically creates template of Javadoc
  - When you type `/**`

Adding Eclipse to Your Menu

- System → Preferences → Main Menu
- Add Eclipse
  - Command: eclipse
  - Click Icon, Browse: /opt/eclipse; select icon
- Add shortcut to top bar
  - In menu, select icon; drag to top bar

Installing at Home

- Go to `www.eclipse.org`
- Select "Downloads"
- Then "Eclipse IDE for Java EE Developers"
  - For developing web applications and other *enterprise* applications

Assignment 5

- Using Eclipse
- Creating an online library
  - 5 classes of objects
  - Driver program
- Due before **Wednesday's class**
  - ***Yes, two class periods away!***
  - But start early