

Objectives

- Polymorphism
 - Dispatch
- Javadocs
- Eclipse

Sept 23, 2011

Sprenkle - CSCI209

1

Review

- When should we make method **static**?
- 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?

Sept 23, 2011

Sprenkle - CSCI209

2

Code Review

- What were the datatypes of your Birthday class's instance variables? Why?
- 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");
    }
}
```

- How could the method be improved?

Sept 23, 2011

Sprenkle - CSCI209

3

Code Review: Good Use of switch Statement

```
public Birthday() {
    int x = random.nextInt(12);
    switch (x) {
        case 1:
            randDay = random.nextInt(29) + 1;
            break;
        case 3:
        case 5:
        case 8:
        case 10:
            randDay = random.nextInt(30) + 1;
            break;
        default:
            randDay = random.nextInt(31) + 1;
            break;
    }
    this.month = months[x];
    this.day = randDay;
}
```

What does this code do?

Sept 23,

4

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

Sept 23, 2011

Sprenkle - CSCI209

5

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 + " ";
}
```

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

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

Sept 23,

Assignment 3 Feedback

- Always use `@Override` annotation
 - Prevents accidental changes to method signature, which would mean that you're not actually overriding the method
- Always document formatting for `toString` and how determining equivalence for `equals`

Sept 23, 2011

Sprenkle - CSCI209

7

POLYMORPHISM & DISPATCH

Sept 23, 2011

Sprenkle - CSCI209

8

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`, `Rooster`, `Hen`, or any class that *inherits from* `Chicken`

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

We can guess the actual types
But compiler can't

Sept 23, 2011

Sprenkle - CSCI209

9

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

Sept 23, 2011

Sprenkle - CSCI209

10

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.

→ Object variable determines how compiler sees object.

- We cannot assign a parent class object to a derived class object variable
 - Ex: `Rooster` is a `Chicken`, but a `Chicken` is not necessarily a `Rooster`

~~`Rooster r = chicken;`~~

Sept 23, 2011

Sprenkle - CSCI209

11

Polymorphism

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

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

Sept 23, 2011

Sprenkle - CSCI209

12

Polymorphism

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

Sept 23, 2011

Sprenkle - CSCI209

13

Dynamic vs. Static Dispatch

- Dynamic dispatch is not necessarily a property of 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

Sept 23, 2011

Sprenkle - CSCI209

14

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?
What happens in execution?

- 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!

Sept 23, 2011

Sprenkle - CSCI209

15

What Will This Code Output?

```
class Parent {
    public Parent() {}
    public void method1() {
        System.out.println("Parent: method1");
    }
    public void method2() {
        System.out.println("Parent: method2");
    }
}

class Child extends Parent {
    public Child() {}
    public void method1() {
        System.out.println("Child: method1");
    }
    public void method2() {
        System.out.println("Child: method2");
    }
}

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

Sept 23, 2011

Review: Inheritance Rules: Access Modifiers

Access modifiers in child classes

- Can make access to child class **less** restrictive but not more restrictive

- Why?**
- What would happen if a method in the parent class is **public** but the child class's method is **private**

Sept 23, 2011

Sprenkle - CSCI209

17

Review: 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

Sept 23, 2011

Sprenkle - CSCI209

18

CASTING

Sept 23, 2011

Sprenkle - CSCI209

19

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

Sept 23, 2011

Sprenkle - CSCI209

20

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];
```

Sept 23, 2011

Sprenkle - CSCI209

21

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];
// Run-time ERROR; chickens[0] 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

Sept 23, 2011

Sprenkle - CSCI209

22

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

Sept 23, 2011

Sprenkle - CSCI209

23

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

Sept 23, 2011

Sprenkle - CSCI209

24

JAVADOCS

"Documentation is a love letter that you write to your future self." – Damian Conway

Sept 23, 2011

Sprenkle - CSCI209

25

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

Sept 23, 2011

Sprenkle - CSCI209

26

Javadoc

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

@param <paramname> <description>

@return <description> (include special cases)

```

startsWith
public boolean startsWith(String prefix)
    Tests if this string starts with the specified prefix.

    Parameters:
        prefix - the prefix.
    Returns:
        true if the character sequence represented by the argument is a prefix of the character sequence
        represented by this string; false otherwise. Note also that true will be returned if the argument is an
        empty string or is equal to this String object as determined by the equals(Object) method.
  
```

Sept 23, 2011

Sprenkle - CSCI209

27

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
 */
  
```

Sept 23, 2011

Expect these types of comments on all methods from now on

Generating Javadocs

- From command-line:


```
javac [options] [packagenames]
      [sourcefiles] [@files]
```
- Or, using Eclipse ...

Sept 23, 2011

Sprenkle - CSCI209


29



Sept 23, 2011

Sprenkle - CSCI209

30



<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

Sept 23, 2011 Sprenkle - CSCI209 31

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
 - Often hidden in GUI

Sept 23, 2011

Sprenkle - CSCI209

32

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 **/****
- Autocompletion of variables, methods
- Formatting code ...
- Shows unused fields/variables
- Shows compiler errors
- ...

Sept 23, 2011

Sprenkle - CSCI209

33

Eclipse Demo

Why can Eclipse provide this functionality? (but Idle can't?)

- Show Birthday class
 - Override **equals** and **toString** methods
- Create a new class
 - Generate Main method, Comments
 - Create a String object, see methods used
- Demonstrate refactoring
 - Rename a field
 - Extract a method (month name)
- Run the Birthday Class (main)
 - Command line arguments

Sept 23, 2011

Sprenkle - CSCI209

34

Your Eclipse Practice

- Start up Eclipse
- Create a new Java project: Assign5
- Create a new Java class: Test
 - Checkbox: Main method, comments
 - Add an instance field: `private int myVar;`
 - Use Source menu to generate a constructor
 - Use Source menu to generate **toString**

Sept 23, 2011

Sprenkle - CSCI209

35

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

Sept 23, 2011

Sprenkle - CSCI209

36

Assignment 5

- Using Eclipse
- Creating an online library
 - 5 classes of objects
 - Driver program
- Could be tedious without IDE
- Due before **Wednesday's class**
 - *Yes, two class periods away!*
 - But start early