Why Object-Oriented?

- Machine code
- Assembly
- Functions/Methods/Structures

1

Noticed?

- Redundancy in writing the Combat classes for lab 3?
- Wouldn't we like to get rid of this…

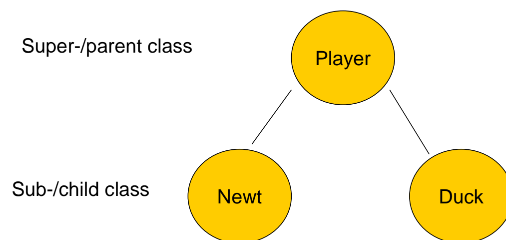
2

Welcome to Inheritance

- The idea is to take all the code that's the same between similar classes (like the Pond Wars characters) and put it into a super class so that all subclasses can then use the common code

3

You've Already Seen This



4

Newt.java

- Classes:
  - http://www.cs.rit.edu/~cs1/lectureNotes/oldCombat/Player.java
  - http://www.cs.rit.edu/~cs1/lectureNotes/oldCombat/Newt.java
- Use "extends" keyword to link super-/sub-classes together
- What should really be in Player?

5

Player Class Contents

- Constants that all players have that are the same
- Variables that all players have (all objects will still get their own instance)
- Methods that are the same for all players

6

Constants

```
public static final int INTELLIGENCE = 0;
public static final int AGILITY = 1;
public static final int STRENGTH = 2;
public static final int NORMAL_ATK = 0;
public static final int SPECIAL_1 = 1;
public static final int SPECIAL_2 = 2;
public static final double BASE_HP = 80.0;
public static final double BASE_FORCE = 1.5;
public static final double BASE_DEFENSE = 3.5;
public static final double BASE_WISDOM = 0.0;
public static final double BASE_STUBBORN = 0.5;
public static final double BASE_DEX = 6.0;
public static final double BASE_DODGE = 4.0;
public static final String CHAR_CLASS = "Newt";
public static final int CLASS_FOCUS = AGILITY;
```

7

More Constants

```
public static final String ATK_0_NAME = "SLIMY PAW";
public static final String ATK_1_NAME = "LICK";
public static final String ATK_2_NAME = "SURE SHOT";
public static final int ATK_0_FORM = AGILITY;
public static final int ATK_1_FORM = AGILITY;
public static final int ATK_2_FORM = AGILITY;
public static final double ATK_1_CHANCE = 70.0;
public static final double ATK_2_CHANCE = 90.0;
```

8

Player

```
public class Player {
public static final int INTELLIGENCE = 0;
public static final int AGILITY = 1;
public static final int STRENGTH = 2;
public static final int NORMAL_ATK = 0;
public static final int SPECIAL_1 = 1;
public static final int SPECIAL_2 = 2;
public static final double BASE_HP = 80.0;
public static final double BASE_FORCE = 1.5;
public static final double BASE_DEFENSE = 3.5;

Etc.
```

9

In Newt

- How do I access those constants?
  - The same way you would if they were in your class: `if (stats == INTELLIGENCE)`
  - You can also access them directly using the term “super”: `if (stats == super.INTELLIGENCE)`
  - This means that you could (technically) have a variable in the subclass that is the same name as the one in the superclass and still be able to access both

10

What about?

- How do I access a variable (say INTELLIGENCE) that is in a superclass of a superclass??? (say Entity is a superclass of Player which is a superclass of Newt)

11

Answer

- Entity.INTELLIGENCE

- What if the variable is not static???

12

Answer

- ((Entity)this).INTELLIGENCE
- Casting the class to the version of a variable you want works, but is not good coding style

13

What else?

- We can move all the instance variables as well as the constants into Player
- Why can't we access them if we move them directly over?

14

Privacy

- If a variable is declared "private" then it can only be accessed by objects inside its own class
- Its children cannot access it
- But you want to use it!

15

Privacy Solutions in Java

- Make the variable "protected" so that it can be used within its class as well as by its subclasses – ok in small projects
- Use accessors/mutators to set/get the variables in the superclass – better for larger projects

```
protected int attackSelected = NORMAL_ATTACK;
```

16

What else?

- All the methods except the constructor are really the same, but use different variable values
- We can move all the methods up into the Player class and set up the various variable values in the constructor of the subclass
- NOTE: this means some of those variables should no longer be final

17

Changing final Variables

```
protected int normalForm = DEXTERITY;
protected int special1Form = DEXTERITY;
protected int special2Form = DEXTERITY;
protected int attackStatOfSelected = normalForm;
protected String normalAttackName = "generic me to death";
protected String attackSpecial1Name = "nope";
protected String attackSpecial2Name = "double dose of nope";
protected String name = "Generic";
```

18

What about private methods?

- **`private double calculateHitPoints()`**
  - Needed by the subclass in the constructor
- **`private double getRange()`**
  - Only used by Player now
- **`private double getBaseDamage()`**
  - Only used by Player now
- **`private double getAttackBase( Player opponent )`**
  - Only used by Player now
- **`private void selectAttack()`**
  - Only used by Player now

19

Private methods

- We can make them all protected
- We can make only the ones used by a subclass now protected

20

Newt: Constructor

```
public Newt() {
   super();
   // Calculate hit points
   currentHitPoints =  calculateHitPoints();

   // character specific variables
   name = "NEWTY";
   normalForm = DEXTERITY;
   special1Form = DEXTERITY;
   special2Form = DEXTERITY;
   attackStatOfSelected = normalForm;
   normalAttackName = "SLIMY PAW";
   attackSpecial1Name = "LICK";
   attackSpecial2Name = "SURE SHOT";
 }
```

21

super

- It's used like the keyword “this” for constructors to call the constructor of your superclass you want to call
- It must be the first line in the constructor for the subclass
- It can have arguments just like “this”
  - Examples: super(), super(name)

22

New Versions

- Classes:
  - http://www.cs.rit.edu/~cs1/lectureNotes/newCombat/Player.java
  - http://www.cs.rit.edu/~cs1/lectureNotes/newCombat/Newt.java

23

What if???

- What if I wanted to make every character have its own attack method?

24

Method Overriding

- You can write a method in the subclass (Newt) that looks like the method in the superclass (Player), but does something different
- You cannot use the method from the superclass once it has been overridden in the subclass

```
public void attack( Player opponent ){
      // Inform opponent of damage
     opponent.getsHit( 10.0 );
}
```

25

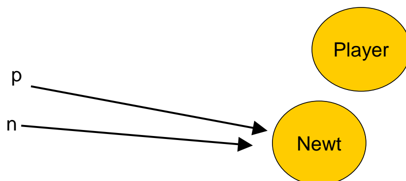
The Mystery of Polymorphism

- Means: many forms
- A reason to read your book!
- The idea:
  - I can create a Newt object which is of super-type Player
  - Therefore in code I can refer to it and assign it to type Player
  - Java takes care of figuring out that we are actually referring to the Newt method for attack as opposed to the Player method as the object is still really a Newt

26

Example

Newt n = new Newt(); // has Newt attack method
Player p = new Player(); // has Player attack method
p = n; // will now call newt's attack method and info



27

From PondModel.java

```
public boolean attack( String name1, String name2 ) {
         Player attacker1 = getPlayerFromName( name1 );
         Player attacker2 = getPlayerFromName( name2 );
         boolean success = true;

         if (attacker1 != null && attacker2 != null) {
           // we want them to attack each other!
           attacker1.attack(attacker2);
           attacker2.attack(attacker1);
         } else success = false;

         return success;
}
```

28

Mysterious Method in Player.java

- It's still outside the scope of this course, but basically loads a class file at run time when given the name of the class file (like Newt)
- It creates the Newt and then assigns it to a player item and returns the player item

```
public static Player getAPlayer( String playerClassName )
```

29

abstract

- Used for classes/methods that are not implemented, but should be implemented in all subclasses
- It's like a contract that makes sure they're implemented
  - http://www.cs.rit.edu/~cs1/lectureNotes/oldCombat/Player.java

30

Inheritance

- What things should be abstract?
- What things should be inherited?
- What things have an "is-a" relationship?
- What things have a "has-a" relationship?

31

Java Data Structures

- Let's go through a real example of a data structure created from an array
  - Java Data Structures: Called the "Collection classes" since they hold and organize collections of data
  - Vector: a collection class that gives nice methods for accessing/storing into arrays
  - Some things inside the class are good and some are not.

32

Games?

- Games need to be careful of how they use memory
  - Game developers may created their own data structures
  - If not, they will figure out the optimal data structure to use for the job
  - They will code to increase performance

33

Speed and Java

- Java with just-in-time compiling isn't too bad, but your code can make a big difference in how things run

34

Creating too many new objects

- Your program can be made to run faster
  - Don't create "new" objects except where really needed
  - Example (pseudo code):

```
String line, allData;
while ( not at the end of data file ) {
        line = read line;
        allData = allData + line;
}
```

35

Solution

- Use StringBuffer as it doesn't create new objects all the time

```
StringBuffer allData;
while ( not at the end of data file ) {
        allData.append( read line );
}
```

36

Data

- Avoid reallocating any arrays where unnecessary even when they are inside of other data structures

Example:
http://www.cs.rit.edu/~cs1/lectureNotes/SillyVector.java

Solution

- http://www.cs.rit.edu/~cs1/lectureNotes/SillyVectorCreated.java
- You should also specify the increment when you aren't sure of the final size so that the array doesn't always double in size.