Today -- Details of Creating Classes

• From requirements to classes
  • Creating a class that will simulate a number game
  • Practice going from requirements to class definitions and fields declarations
  • Random number generators
  • Using an import statement to allow you to use a short name

• Creating Classes

• Hierarchy
Simulating a Number Guess Game

- Have you ever played the pick a number from 1-10 or 1-100 game?
  - One person randomly picks a number in the specified range
  - The other person guesses a number
  - The person who picked the number replies
    - Higher if the guess is too low
    - Lower if the guess is too high
    - Or that is right if the guess is the picked number
      - You could also track the number of guesses
What do we need?

• We need a new class. For each game we will create a new object of this class.
  – It will need to know the minimum number that it can pick from
  – It will need to know the maximum number that it can pick from
  – It will need to pick a number in that range
  – It will need to ask the user for a guess and compare the guess to the picked number
    • And reply higher, lower, or that is right when the guess matches the picked number
  – It will need to keep track of how many guesses have been made
Going from Specifications to a Class

• What should the name of the class be?
  – Class names should be singular
  – Class names should be an indicator of what objects of the class are for or can do
  – Class names start with an uppercase letter and uppercase the first letter of each new word

• Any ideas?
Create the NumberGuessGame Class

• Start by creating a new class in DrJava
  – Select (Untitled) in the Files pane or
  – Click on the “New” button in the menu bar
• Type the following in the definitions pane in DrJava
  ```java
  public class NumberGuessGame
  {
  }
  ```
• Save it in NumberGuessGame.java
• Compile it using the “Compile All” button
  – Which creates NumberGuessGame.class
Going from Specifications to Fields

• What fields (state) does each object of the NumberGuessGame need to have?
  – What names should we use for these?
  – What should the types be for the fields?

• Requirements
  – minimum number that it can pick from
  – maximum number that it can pick to
  – Picked number for a game
  – Track how many guesses have been made
Add Field Declarations

• Declaring fields
  – Each field should be private
  – You can assign a value to a field when you declare it

• Each number guess game should have
  – A minimum number
    
    ```java
    private int minNumber = 1;
    ```
  – A maximum number
    
    ```java
    private int maxNumber = 100;
    ```
  – A picked number
    
    ```java
    private int pickedNumber;
    ```
  – A number of guesses so far
    
    ```java
    private int numGuesses = 0;
    ```
Add the Fields

• Edit NumberGuessGame and add the fields

```java
public class NumberGuessGame
{
    // fields (data) 
    private int minNumber = 1;
    private int maxNumber = 100;
    private int pickedNumber;
    private int numGuesses = 0;
}
```
Picking a Random Number

- There is a class in Java that allows you to pick a pseudo random number
  - `java.util.Random`
  - You will want to import this class to use the short name
    ```java
    import java.util.Random; // before the class definition
    ```
- You can create an object of this class
  ```java
  Random randomNumGen = new Random();
  ```
- You can get a random number from 0 to one less than a specified integer
  ```java
  int randomNum = randomNumGen.nextInt(specifiedInt);
  ```
Picking from a Min to a Max

• If the Random class returns from 0 to 1 less than a specified integer
  – How do we pick from the minimum to the maximum?
    • No matter what the minimum and maximum are?
  – To pick a number from 1 to 10
    • This is 10 values
      – so pick from 0 to 10 (returns 0 to 9)
      – And add 1 to the result (results in 1 to 10)
  – To pick a number from 11 to 15
    • This is 5 values
      – So pick from 0 to 5 (returns 0 to 4)
      – Add 11 to the result (results in 11 to 15)
  – To pick in any range
    ```
    int numValues = this.maxNumber - this.minNumber + 1;
    this.pickedNumber = this.randomNumGen(numValues);
    this.pickedNumber = this.pickedNumber + this.minNumber;
    ```

Based on slides by Barb Ericson, Georgia Institute of Technology
Add an Import and a Field

- Add the import statement before the class declaration
  ```java
  import java.util.Random;
  public class NumberGuessGame
  ```

- Add the new field for the random number generator
  ```java
  private int numGuesses = 0;
  private Random randomNumGen = new Random();
  ```
Creating Constructors

• Constructors actually initialize the new object
  – Usually set field values for the object
• If the user doesn’t specify a min and max number
  – Use the defaults and pick a random number between the min and max
• Add another constructor that let’s the user specify the min and max
Declaring Constructors

• To declare a constructor
  – Specify the visibility and the name of the class followed by a parameter list
    ```
    public ClassName(parameterList)
    ```

• You can declare more than one constructor
  – As long as the parameter lists are different
    ```
    public NumberGuessGame()
    public NumberGuessGame(int min, int max)
    ```
No Argument Constructor

/**
 * Constructor that takes no parameters
 * It uses the default min and max
 */

public NumberGuessGame()
{
    int numValues = this.maxNumber - this.minNumber + 1;
    this.pickedNumber =
        this.randomNumGen.nextInt(numValues);
    this.pickedNumber = this.pickedNumber +
        this.minNumber;
}
Constructor with a Min and Max

```java
/**
 * Constructor that takes a min and max
 * It uses the passed min and max
 * @param min the minimum number in the range
 * @param max the maximum number in the range
 */

public NumberGuessGame(int min, int max)
{
    this.minNumber = min;
    this.maxNumber = max;
    int numValues = this.maxNumber - this.minNumber + 1;
    this.pickedNumber = this.randomNumGen.nextInt(numValues);
    this.pickedNumber = this.pickedNumber + this.minNumber;
}
```

Based on slides by Barb Ericson,
Georgia Institute of Technology
Summary

• To look for classes
  – Underline nouns
    • Nouns with several pieces of data associated with them are classes
• First determine the classes you will need and create them
• Then determine the data each object of that class will need
  – And declare fields
• The `Random` class in package `java.util`
  – Can be used to pick a random number
• You can use an import statement
  – To let you use a short name for a class that isn’t in `java.lang`
Today -- Details of Creating Classes

• From requirements to classes

• Methods
  – Pulling out a method
    • That is called by the constructors
  – Getting input
    • Using SimpleInput class from Georgia Tech
  – Showing output
    • Using SimpleOutput class from Georgia Tech
  – Generating random sentences

• Hierarchy
Pull out a Method

- Both Constructors need to pick a random number using the minimum and maximum
- We can pull out this common code and make a method for it

```java
public void pickNumber()
```

- And then call the method in each constructor
Pick a Number Method

```java
public NumberGuessGame(int min, int max) {
    this.minNumber = min;
    this.maxNumber = max;
    this.pickNumber();
}

public void pickNumber() {
    int numValues = this.maxNumber - this.minNumber + 1;
    this.pickedNumber = this.randomNumGen.nextInt(numValues);
    this.pickedNumber = this.pickedNumber + this.minNumber;
}
```
Need a Method to Play the Game

• Set a variable to not done
• Loop while not done
  – Get the current guess
  – Increment the number of guesses
  – Check if the guess was right
    • If so tell the user the guess was right and how many guesses it took
    • Set a variable (done) to stop the loop
  – Check if the guess was low
    • Tell the user
  – Else the guess was too high
    • Tell the user
Need a Way to Interact with User

• Use the SimpleInput class for input
  – Created by Georgia Tech
  – Has a method `getIntNumber(String message)`
    • That will display the message in a pop-up window and return an integer number entered by the user

• Use the SimpleOutput class for output
  – Created by Georgia Tech
  – Has a method `showInformation(String message)` which will display the output in a pop-up window
    • And wait for the user to push a button to show it has been read

Based on slides by Barb Ericson, Georgia Institute of Technology
Going from Algorithm to Code

• Set a variable to not done
• Loop while not done
  – Get the current guess
  – Increment the number of guesses
  – Check if the guess was right
    • If so tell the user the guess was right and how many guesses it took
    • Set a variable (done) to stop the loop
  – Else check if the guess was low
    • Tell the user
  – Else the guess was too high
    • Tell the user

Use boolean done variable and set it to false
Use a while loop that loops as long as the done isn’t true
while (!done)
  Using SimpleInput
    And a variable guess
    numGuesses++;
    if (guess == pickedNumber)
      Use SimpleOutput and numGuesses
      Change done to true
    else if (guess < pickedNumber)
      else
public void playGame()
{
    boolean done = false;

    // loop till guess is correct
    while (!done)
    {
        // need to get a guess from the user
        int guess = SimpleInput.getIntNumber("Pick a number "+
                  "between " + this.minNumber + " and " +
                  this.maxNumber);

        // increment the number of guesses
        this.numGuesses++;

        // we need to check the guess (compare to pickedNum)
        if (guess == this.pickedNumber)
        {
        
        

Based on slides by Barb Ericson,
Georgia Institute of Technology
Play

// tell the user she/he was right
SimpleOutput.showInformation("That was right! " +
    "It took you " +
    this.numGuesses + " tries");

done = true;
}
else if (guess < this.pickedNumber)
{
    // we need to tell the user too low
    SimpleOutput.showInformation("Too low");
}
else
{
    // tell the user the guess is too high
    SimpleOutput.showInformation("Too high");
}
}
Add a main method

```java
public static void main(String[] args)
{
    NumberGuessGame game = new NumberGuessGame();
    game.playGame();
}
```
Random Sentence Generator Exercise

• Write a class that can generate random sentences.
  – Create a class SentenceGenerator
    • That has an array of nouns
    • An array of verbs
    • An array of phrases
    • And a method generate sentence which will return a String object that has a randomly selected noun, verb, and phrase appended together
Summary

• If more than one constructor needs to do the same thing
  – Pull out the common thing and put it in a method
  – And call the method in the constructors

• You can get input from the user
  – Using SimpleInput

• You can display output to the user
  – Using SimpleOutput

• You can use java.util.Random
  – To create random sentences
Today -- Painful Details of Classes

• From requirements to classes
• Methods
• Hierarchy
  – Inheriting from a class
  – The implicit call to super()
  – Calling parent constructors explicitly
  – Overriding a parent method
  – How methods invocations are resolved
Creating an Inherited Class

• Create a class ConfusedTurtle that inherits from the *Turtle* class
  – But when a ConfusedTurtle object is asked to turn left, it should turn right
  – And when a ConfusedTurtle object is asked to turn right, it should turn left

Based on slides by Barb Ericson, Georgia Institute of Technology
Inheriting from a Class

- To inherit from another class
  - Add `extends ClassName` to the class declaration

```java
public class ConfusedTurtle extends Turtle {
}
```

- Save in `ConfusedTurtle.java`
- Try to compile it
Compile Error?

• If you try to compile ConfusedTurtle you will get a compiler error
  – Error: cannot resolve symbol
  – symbol: constructor Turtle()
  – location: class Turtle
• Why do you get this error?
Inherited Constructors

• When one class inherits from another all constructors in the child class will have an implicit call to the no-argument parent constructor as the first line of code in the child constructor
  – Unless an explicit call to a parent constructor is there as the first line of code
    • Using super(paramList);
Why is an Implicit Call to Super Added?

- Fields are inherited from a parent class
  - But fields should be declared private
    - Not public, protected, or package visibility
      - Lose control over field at the class level then
  - But then subclasses can’t access fields directly
  - How do you initialize inherited fields?
    - By calling the parent constructor that initializes them
      - Using super(paramList);
Explanation of the Compile Error

• There are no constructors in ConfusedTurtle
  – So a no-argument one is added for you
    • With a call to super();
  – But, the Turtle class doesn’t have a no-argument constructor
    • All constructors take a world to put the turtle in

• So we need to add a constructor to ConfusedTurtle
  – That takes a world to add the turtle to
    • And call super(theWorld);
public class ConfusedTurtle extends Turtle {

    /**
     * Constructor that takes a world and
     * calls the parent constructor
     * @param theWorld the world to put the
     * confused turtle in
     */

    public ConfusedTurtle(World theWorld) {
        super (theWorld);
    }
}

Based on slides by Barb Ericson,
Georgia Institute of Technology
Add a Constructor that takes a World

```java
public class ConfusedTurtle extends Turtle {
    /**
     * Constructor that takes a world and calls the parent constructor
     * @param modelDisplayObj the world to put the confused turtle in
     */
    public ConfusedTurtle(ModelDisplay modelDisplayObj) {
        super(modelDisplayObj);
    }
}
```

In the book, it goes one more level up the hierarchy from Turtle to SimpleTurtle whose instance variable is the super class of World, called “ModelDisplay”.

Based on slides by Barb Ericson, Georgia Institute of Technology
Try this Out

• Compile ConfusedTurtle
  – It should compile

• Try it out
  – It should act just like a Turtle object

• How do we get it to turn left when asked to turn right?
  – And right when asked to turn left?
    • Use super.turnLeft() and super.turnRight()
      – super is a keyword that means the parent class

What would happen if we used “this.turnLeft()” instead of “super.turnLeft()”? 

Based on slides by Barb Ericson,
Georgia Institute of Technology
Resolving Methods

- When a method is invoked (called) on an object
  - The class that created the object is checked
    - To see if it has the method defined in it
      - If so it is executed
      - Else the parent of the class that created the object is checked for the method
      - And so on until the method is found
  
- Super means start checking with the parent of the class that created the object

Based on slides by Barb Ericson, Georgia Institute of Technology
Polymorphism

- Means many forms
- Allows for processing of an object based on the object’s type
- A method can be declared in a parent class
  - And redefined (overridden) by the subclasses
- Dynamic or run-time binding will make sure the correct method gets run
  - Based on the type of object it was called on at run time
Confused Turtle turnLeft and turnRight

/**
 * Method to turn left (but confused turtles turn right)
 */

public void turnLeft()
{
    super.turnRight();
}

/**
 * Method to turn right (but confused turtles turn left)
 */

public void turnRight()
{
    super.turnLeft();
}
Try out ConfusedTurtle

```java
> World earth = new World();
> Turtle tommy = new Turtle(earth);
> tommy.forward();
> tommy.turnLeft();
> ConfusedTurtle bruce = new ConfusedTurtle(earth);
> bruce.backward();
> bruce.turnLeft();
> bruce.forward();
> tommy.forward();
> tommy.turnRight();
> bruce.turnRight();
```
Override Methods

• Children classes inherit parent methods
  – The confused turtle knows how to go forward and backward
    • Because it inherits these from Turtle
• Children can override parent methods
  – Have a method with the same name and parameter list as a parent method
    • This method will be called instead of the parent method
      – Like turnLeft and turnRight
What is Happening?

• Each time an object is asked to execute a method
  – It first checks the class that created the object to see if the method is defined in that class
  • If it is, it will execute that method
  • If it isn’t, it will next check the parent class of the class that created it
    – And execute that method if one is found
    – If no method with that name and parameter list is found, it will check that class’s parent
      » And keep going till it finds the method
Method Overloading

SimpleTurtle: Class
- forward()
- backward()
- turnLeft()
- turnRight()

Turtle: Class
- drawSquare()

ConfusedTurtle: Class
- turnLeft()
- turnRight()

Obj: Turtle

Obj: ConfusedTurtle

Obj: ConfusedTurtle

Based on slides by Barb Ericson, Georgia Institute of Technology
Exercises

• Create a DizzyTurtle class
  – That turns a bit to the left and goes forward when asked to go forward
  – And turns a bit to the right and goes backward when asked to go backward

• Create a SlowTurtle class
  – That only goes forward and backward by 50 instead of 100 if you don’t tell it how much to go forward or backward

• Create a StubbornTurtle class
  – Has a 50% chance of doing what you ask
Summary

- Use the `extends` keyword to specify the parent class
  - When you declare a class
    ```java
    public class ConfusedTurtle extends Turtle
    ```
- A class inherits methods and fields from a parent class
  - But doesn’t have direct access to private fields and methods
- A implicit call to the no-arg parent constructor will be added
  - If there isn’t a call to `super(paramList)` as the first line of code in a child constructor
- A class can override a parent method
  - To be called instead of the parent method
    - Using the same method name and parameter list as a parent method
- A method can invoked a parent’s method
  - Using `super.methodName(paramList);`
Coming Attractions

• HW 10 - oche
  – due on Thursday
  – look at Python echo recipe
    • what is different?

• Wednesday
  – review for Final
  – take home announced

• Thursday (2-3 PM) in 110 McB
  – Open House
  – Learn about game design, animation, multimedia, cyberart
  – FOOD!