Lecture 22

- Covers
  - Programming with methods
  - Static attributes
  - Static methods
  - The Math class
- Reading: Savitch 5.1, 5.2

Lecture overview

- The this reference
- Static attributes and static methods
- More on the Math class
- More on the main( ) method

- The this reference

The this reference

- We can refer to an attribute or a method of an object from without or from within
- From without, we must specify both the name that refers to the object, and the name of the attribute or method
- When we are referring to an attribute or method in the same object, we refer to the same object with a specially named reference: this

The this reference

- Therefore, we can refer to any attribute in the same object as

  this.<attribute>

- And any method in the same object as

  this.<method>

Omitting this

- When we are referring to an attribute or method in the same object, Java lets us leave out the this
- We can simply refer to an attribute in the same object by

  <attribute>

  when there is no ambiguity (i.e. name conflict)
- And invoke a method in the same object as

  <method>

One common use of this

- We often use this to distinguish the attribute from an argument
- Example

```
public void setHours(int hours)
{
        if( 0 <= hours && hours <= 23)
        {
                this.hours = hours;
        }
        else
        {
                this.hours = 0;
        }
}
```

- Writing methods using helper methods

Handling complex methods

- When a method is too long or too complicated, it may be a good idea to break it up into smaller methods

Example

- Write a program that plays a guessing game with the user
- The program "thinks" of a number between 1 and 100 (inclusive) that the user has to guess
- Each time the user guesses a number that is not correct, the game indicates whether the number they guessed is too high or too low
- After the user has guessed the correct number, the program displays how many guesses were required

Example

I'm thinking of a number between 1 and 100.
Guess a number between 1 and 100 (inclusive): 50
That was too low!
Guess a number between 51 and 100 (inclusive): 12
Silly Duffer! That was way too low!
Guess a number between 51 and 100 (inclusive): 65
That was too high!
Guess a number between 51 and 64 (inclusive): 102
Silly Duffer! That was way too high!
Guess a number between 51 and 64 (inclusive): 58
That was too low!
Guess a number between 59 and 64 (inclusive): 63
That was too high!
Guess a number between 59 and 62 (inclusive): 60
That was too low!
Guess a number between 61 and 62 (inclusive): 61
You guessed 61 which was the right number!
It took you 8 guesses.

Designing a guessing game class

- Each guessing game has the attributes
  - numberToGuess
    - The number that the user "thinks" of
  - numberOfTries
    - The number of guesses the user has had so far
  - upperBound
    - The upper boundary of the current range in which the user should be guessing (starts at 100)
  - lowerBound
    - The lower boundary of the current range in which the user should be guessing (starts at 1)

Designing a guessing game class

- Each guessing game has a constructor that
  - Sets the upper and lower boundary
  - "Thinks" of a number
  - Sets numberOfTries to 0 initially
- Each guessing game has a method to play the game

Define the class header

```
public class GuessingGame
{

}
```

Define the attributes

```
public class GuessingGame
{
   private int numberToGuess;
   private int numberOfTries;
   private int upperBound;
   private int lowerBound;
}
```

Define the constructor

```
public GuessingGame( )
{
    numberToGuess = (int)(Math.random( ) * 100) + 1;
    numberOfTries = 0;
    upperBound = 100;
    lowerBound = 1;
}
```

Math.random( )

- Math.random( ) is a class method of the Math class
- It returns a pseudo-random number (a double) between 0 and 1 (exclusive)
- We can convert this “random” number to an integer between 1 and 100 (inclusive)

Define the methods: playGame()

```
public void playGame( )
{
    int guess;
    boolean correctGuess = false;
    System.out.println("I'm thinking of a number between "
                              + lowerBound + " and " + upperBound + ".");
    do
    {
        System.out.print("Guess a number between " + lowerBound
                               + " and " + upperBound + " (inclusive): ");
        guess = keyboard.nextInt( );
        correctGuess = this.checkGuess(guess);
    } while (!correctGuess);

    System.out.println("You guessed " + guess
                              + " which was the right number!");
    System.out.println("It took you " + numberOfTries + " guesses.");
}
```

Helper methods

- In the playGame( ) method of the GuessingGame class, checking the guess against the numberToGuess is done in a helper method
- We use helper methods to make our programs easier to read and easier to manage
- When defining a helper method, consider whether you want it to be usable by any class or only within the object itself.

Define the methods: checkGuess( )

```
private boolean checkGuess(int guess)
{
    ++numberOfTries;
    if (guess == numberToGuess)
    {
        return true;
    }
    else if (guess < lowerBound)
    {
        System.out.println("Silly Duffer!  That was way too low!");
    }
    else if (guess > upperBound)
    {
        System.out.println("Silly Duffer!  That was way too high!");
    }
```

checkGuess( )

```
    else if (guess < numberToGuess)
    {
        System.out.println("That was too low!");
        lowerBound = guess + 1;
    }
    else
    {
        System.out.println("That was too high!");
        upperBound = guess - 1;
    }
    return false;
}
```

Alternative logic for checkGuess( )

```
if (guess == numberToGuess)
{
	return true;
}
else
{
	if (guess < lowerBound)
	{
		System.out.println("Silly  Duffer!  That was way too low!");
	}
	else if (guess > upperBound)
	{
		System.out.println("Silly  Duffer!  That was way too high!");
	}
	else if (guess < numberToGuess)
	{
		System.out.println("That  was too low!");
		lowerBound = guess + 1;
	}
	else
	{
		System.out.println("That  was too high!");
		upperBound  = guess - 1;
	}
	return false;
}
```

Making helper methods private

- We only want the checkGuess method to be invoked by the method playGame which is in the same object
- We therefore restrict the access to checkGuess by making it private

- Static attributes and static methods

Extension

- Suppose we are going to write a program to play the game several times, and we want to know the average number of guesses we made per game (for that run of the program)
- We can do that by having
  - an attribute to count the number of games we play and
  - an attribute to count the total number of guesses we make

Extension

- These two attributes are not information about any particular game instance. They are information about the class. They are class (or static) attributes
- We also need methods to update these attributes and to compute the average number of guesses. Once again, they do not belong to any particular object. They belong to the class. They are class (static) methods

Static methods

- Static attributes and methods are defined with the keyword static
- Static attributes and methods are properties of the class, not of the individual objects
- In particular, static attributes are shared by all instances of the class
  - They cannot differ from instance to instance
- Static methods implement the notion of class methods

Static attributes

- The value of a static attribute can be changed by class methods (static) or by instance methods (non-static)
- Static attributes are shared by the class and all its instances
- When we create a new instance of a class, it does not get its own copy of that static attribute but shares it with the class and its other objects

Example

- Suppose we want the GuessingGame class to keep track of how many times the game has been played (irrespective of which objects have had the game played on them)
- We can give the GuessingGame class a static attribute to keep track of the number of times playGame( ) has been called on any object

Example

```
public class GuessingGame
{
    // instance attributes would go here
    private static int numberOfGames = 0;

    public void playGame( )
    {
        ++numberOfGames;
        …
    }

    public static int getNumberOfGamesPlayed( )
    {
        return numberOfGames;
    }
```

Example

```
public static void main(String[ ] args)
{
    GuessingGame trialGame1 = new GuessingGame( );
    trialGame1.playGame( );
    GuessingGame trialGame2 = new GuessingGame( );
    trialGame2.playGame( );
    GuessingGame trialGame3 = new GuessingGame( );
    trialGame3.playGame( );
    System.out.println("You played " +
              GuessingGame.getNumberOfGamesPlayed( ) +
              " games");
}
```

** Because getNumberOfGamesPlayed is a static method of the same class we could call it without the GuessingGame.*

- More on static attributes and the Math class

Static attributes in memory

trialGame1 | 123D

123D

numberToGuess:
numberOfTries:
upperBound:
lowerBound:

**Each* GuessingGame *object has its own copy of the instance attributes*

**Only one copy of the class (static) attributes is kept*

GuessingGames.numberOfGames | 0

Usage

- There are three common uses of static properties
  1. To define commonly used constants
  2. To define methods that are independent of specific objects (usually called utilities)
  3. To define shared values (shared by all instances of the class)

Examples

- PI is defined as a static constant in the Math class
- random( ) is defined as a static method in the Math class

Static attributes

- Constants are usually defined as static attributes
- A static attribute belongs to the class and is shared by all instances of the class
- Constants may be made public or private
- In some cases, we wish our class attribute to be usable only by the class or its instance objects (e.g. numberOfGames) and therefore it is made private

Static attributes

- In some cases, we wish our class attribute to usable by any class or its instance objects (e.g. PI) and therefore it is made public
- Java's Math class defines the constants PI and E as publicly available static constants
- Constants are indicated by the keyword final

```
public final static double PI = 3.14159;
```

The Math class

- In package java.lang
  - A package is a collection of classes
- Provides common mathematical constants and functions
- Selected constants and functions

```
public final static double PI
public final static double E
public static int abs(int num)
public static int round(double num)
public static double sqrt(double num)
public static double pow(double n1, double n2)
```

The Math class

public static double sin(double angle)
public static double cos(double angle)
public static double tan(double angle)
public static double asin(double num)
public static double acos(double num)
public static double atan(double num)
public static double log(double num)
public static double exp(double num)
public static double random( )

- Notes on the main method

Testing a class

- Frequently we want to test a class separately
- To do that, we need to create a main method in a class which will instantiate an object of that class and try out its methods
- We can create a separate tester class and put a main method there (recommended for clarity)
- Alternatively, we can and put a main method into the class (the one to be tested) itself

Another look at a main() method

```
public static void main(String[ ] args)
{
    GuessingGame trialGame = new GuessingGame( );
    trialGame.playGame( );
}
```

main() as a static method

- When we start a program, there are no instance objects, only classes defined
- Therefore main must be declared as a static method
- If we wish to invoke non-static methods from a static method (e.g. testing playGame( ) from main( )), we would have to create an instance of the class to invoke the instance methods on

Next lecture

- Designing methods
- Procedural abstraction, top-down design
- Drivers and stubs
- Testing
- Formatting decimal output