Lecture 24

- Covers
  - Overloading methods
  - Automatic type conversion
- Reading: Savitch 5.4

Lecture overview

- Method overloading
- Automatic type conversion

- Method overloading

Method Overloading

- Method overloading is the situation in which two or more methods in the same class have the same name
- In such situations, the overloaded methods, though having the same name, must have different signatures
  - The signature of a method is made up of the method name and the sequence of its parameter types
  - Return types, access modes, and whether the methods are static or non-static are not part of the signature

Method Overloading

- Thus, the overloaded methods, though having the same name, must have different sequences of parameter types

Examples

```
public class A
{
        private  void f( ) { }
        private  void f( int i ) { }
        private  void f( double i) { }
}
```

valid?

Examples

```
public class B
{
        private  void  f( int i ) { }
        private  int f( int i ) { }
}
```

valid?

Examples

```
public class C
{
        private void f( int i ) { }
        private void f( int i, int n ) { }
}
```

valid?

Examples

```
public class D
{
        private void f( int i, double d ) { }
        private void f( int n, double x ) { }
}
```

valid?

Examples

```
public class E
{
        private void f( double d, int i ) { }
        private void f( int i, double d ) { }
}
```

valid?

Another Example

- Write a QuoteGenerator class that outputs a famous quote to the screen
- Methods should be provided to display a quote
  - Specified by number
  - Specified by author
  - Randomly

Example – define the class

```
public class QuoteGenerator
{

}
```

Example – define the attributes

```
public class QuoteGenerator
{
   private static final String quote1 = "Everything should be made as " +
                    "simple as possible, but not one bit simpler";
   private static final String quote2 = "Eye for an Eye will make the" +
                    " whole world blind";
   private static final String quote3 = "Those parts of the system " +
                    "that you can hit with a hammer (not advised)\n" +
                    "are called hardware; those program instructions " +
                    "that you can only curse at\nare called software.";
   private static final String quote4 = "640K ought to be enough for anybody";
   private static final String quote5 = "Computers are useless. They " +
                    "can only give you answers";
   private static final String quote6 = "If you have any trouble " +
                    "sounding condescending, find a Unix user\nto " +
                    "show you how it's done.";
   private static final String quote7 = "There is no reason for any " +
                    "individual to have a computer in his home";
   private static final String quote8 = "There are two major products " +
                    "that come out of Berkeley: LSD and UNIX.\nWe " +
                    "don't believe this to be a coincidence";
}
```

Example - define the methods

- *Method to return a quote based on the quote number*

```
public static void displayQuote(int quoteNumber)
{
    switch (quoteNumber)
    {
        case 1: System.out.println("\"" + quote1 + "\""); break;
        case 2: System.out.println("\"" + quote2 + "\""); break;
        case 3: System.out.println("\"" + quote3 + "\""); break;
        case 4: System.out.println("\"" + quote4 + "\""); break;
        case 5: System.out.println("\"" + quote5 + "\""); break;
        case 6: System.out.println("\"" + quote6 + "\""); break;
        case 7: System.out.println("\"" + quote7 + "\""); break;
        case 8: System.out.println("\"" + quote8 + "\""); break;
        default: System.out.println("\"There are only two truly " +
                    "infinite things, the universe and stupidity.\n" +
                    " And I am unsure about the universe.\"");
    }
}
```

- *Method to return a quote based on the author*

```
public static void displayQuote(String author)
{
    if (author.equalsIgnoreCase("einstein"))
    {
        System.out.println("\"" + quote1 + "\"");
    }
    else if (author.equalsIgnoreCase("ghandi"))
    {
        System.out.println("\"" + quote2 + "\"");
    }
    else if (author.equalsIgnoreCase("anonymous"))
    {
        System.out.println("\"" + quote3 + "\"");
    }
    else if (author.equalsIgnoreCase("gates"))
    {
        System.out.println("\"" + quote4 + "\"");
    }
    else if (author.equalsIgnoreCase("picasso"))
    {
        System.out.println("\"" + quote5 + "\"");
    }
```

- *Method to return a quote based on the author*

```
else if (author.equalsIgnoreCase("adams"))
{
    System.out.println("\"" + quote6 + "\"");
}
else if (author.equalsIgnoreCase("olson"))
{
    System.out.println("\"" + quote7 + "\"");
}
else if (author.equalsIgnoreCase("anderson"))
{
    System.out.println("\"" + quote8 + "\"");
}
else
{
    System.out.println("\"There are only two truly " +
            "infinite things, the universe and stupidity.\n" +
            " And I am unsure about the universe.\"");
}
}
```

Method overloading

- Both the method to display a quote based on quote number and the method to display a quote based on author have the same name
- Can we have more than one method with the same name in the same class?
- Method overloading allows us to do this, as long as the methods' signatures are different

Method overloading

- A method's signature is the combination of the method's name and the number and type of its parameters
- In the previous example, these methods have different signatures

```
public static void displayQuote(int quoteNumber)
public static void displayQuote(String author)
```

- One method expects an integer argument, the other a String argument

Example

```
public static void main(String[ ] args)
{
    Scanner keyboard = new Scanner(System.in);
    int quoteNumber;
    System.out.print("Enter the number of the quote you wish to " +
                              "see\n [1-8]: ");
    quoteNumber = keyboard.nextInt( );
    displayQuote(quoteNumber);
    keyboard.next( ) // gets rid of end of line character after int
    String author;
    System.out.println("Enter the author of the quote you wish to " +
                              "see\n[einstein, ghandi, anonymous, gates, " +
                              "picasso, adams, olson, anderson]: ");
    author = keyboard.nextLine( );
    displayQuote(author);
}
```

Method overloading

- We have already been using overloaded methods

```
println( );
println("Hello");
println(5.5);
```

Example

- Now we want to write a third method to display a random quote
- This method will take no arguments

```
public static void displayQuote( )
```

- This method, again, has a different signature

Example

```
public static void displayQuote( )
{
    int quoteNumber = (int) (Math.random( ) * 9);
    switch (quoteNumber)
    {
        case 1: System.out.println("\"" + quote1 + "\""); break;
        case 2: System.out.println("\"" + quote2 + "\""); break;
        case 3: System.out.println("\"" + quote3 + "\""); break;
        case 4: System.out.println("\"" + quote4 + "\""); break;
        case 5: System.out.println("\"" + quote5 + "\""); break;
        case 6: System.out.println("\"" + quote6 + "\""); break;
        case 7: System.out.println("\"" + quote7 + "\""); break;
        case 8: System.out.println("\"" + quote8 + "\""); break;
        default: System.out.println("\"There are only two truly " +
                    "infinite things, the universe and stupidity.\n" +
                    " And I am unsure about the universe.\"");
    }
}
```

Example

- The method to generate a random quote repeats a lot of code from the first displayQuote method written
- We can invoke that first method in the random quote method, rather than repeating the code

Example

```
public static void displayQuote( )
{
    int quoteNumber = (int) (Math.random( ) * 9);
    displayQuote(quoteNumber);
}
```

Restrictions on overloading

- Overloading is permitted only when the type or number of the parameters are different
- We could not have two methods that differed only in their return type
- We could not have two methods that differed only by one being static and the other non-static

- Automatic type conversion

int to double automatic type conversion

- When a method expects a parameter of type double, but we give it a value of type int, it will automatically convert the int value into double

Example

What is output?

```
public class ATCDemo1
{
        private static double change( double d )
        {
                return  2 * d ;
        }
}

// test
public static void main(String [ ] args)
{
        System.out.println( ATCDemo1.change( 5 );
}
```

Example

What is output?

```
public class ATCDemo1
{
        private static double change( double d )
        {       return 2 * d;  }

        private static int change( int i )
        {       return 3 * i ;  }
}

// test
public static void main(String [ ] args)
{
        System.out.println( ATCDemo1.change( 5 );
}
```

Example

- The example shows that Java uses an overloaded method before trying to do automatic type conversion

Class exercise

- Write a utility class that defines overloaded methods to
  - Calculate the maximum of two integers
  - Calculate the maximum of three integers
  - Calculate the maximum of two doubles
  - Calculate the maximum of three doubles

Automatic type conversion

- If we had only defined the maximum methods for doubles and we had called a maximum method as follows

```
maximum(3, 6);
```

it would use the maximum method that took two doubles… why?

Automatic type conversion

- It would automatically convert the two integers into doubles
- This is referred to as automatic type conversion

Automatic type conversion and overloading

- Given all four maximum methods, the call

  ```
  maximum(3, 6);
  ```

  would use the method that took two ints as arguments
- Java uses an overloaded method before trying to do automatic type conversion

Class exercise

- Given the DigitalClock class from previous lectures, overload the tick( ) method so that it takes an integer parameter, the number of minutes to tick over
- Assume the parameter is a positive integer

```
DigitalClock dc = new DigitalClock( );
dc.tick( );
dc.tick(25);
```

Reminder

```
public class DigitalClock
{
    private int hours;
    private int minutes;
    …
    public void tick( )
    {
        minutes++;
        if (minutes == 60)
        {
            minutes = 0;
            hours++;
        }
        if (hours == 24)
        {
            hours = 0;
        }
    }
}
```

Class exercise

- Write a utility class that defines overloaded methods to calculate the perimeter of the following plane figures
  - A square given its side length (an integer)
  - A circle given its radius (a double)
  - A rectangle given its length and breadth (2 doubles)
- Write a driver to test your class

Next lecture

- Constructors
- More information hiding
- Packages