

cs3101-003 Java: lecture #2

- news:
 - NEW Class web page:
<http://www.cs.columbia.edu/~cs3101>
 - homework #1 due today
 - homework #2 out today
- today's topics:
 - branching with the `Switch` statement
 - looping (counter-controlled loops)
 - native classes
 - more looping (condition-controlled loops)
 - inheritance

branching with `switch` (1): recall `if...`

```
public class ex2a {
    public static void main ( String[] args ) {
        int i = (Integer.valueOf( args[0] )).intValue();
        if ( i == 1 ) {
            System.out.println( "one, two, buckle my shoe" );
        }
        else if ( i == 3 ) {
            System.out.println( "three, four, shut the door" );
        }
        else if ( i == 5 ) {
            System.out.println( "five, six, pick up sticks" );
        }
        else if ( i == 7 ) {
            System.out.println( "seven, eight, lay them straight" );
        }
        else if ( i == 9 ) {
            System.out.println( "nine, ten, a big fat hen" );
        } // end if-else
    } // end main()
} // end of class ex2a
```

branching with `switch` (2): simple statements.

```
public class ex2b {
    public static void main ( String[] args ) {
        int i = (Integer.valueOf( args[0] )).intValue();
        switch( i ) {
            case 1:
                System.out.println( "one, two, buckle my shoe" );
                break;
            case 3:
                System.out.println( "three, four, shut the door" );
                break;
            case 5:
                System.out.println( "five, six, pick up sticks" );
                break;
            case 7:
                System.out.println( "seven, eight, lay them straight" );
                break;
            case 9:
                System.out.println( "nine, ten, a big fat hen" );
                break;
        } // end switch
    } // end main()
} // end of class ex2b
```

branching with `switch` (3): compound statements.

```
public class ex2c {
    public static void main ( String[] args ) {
        int i = (Integer.valueOf( args[0] )).intValue();
        switch( i ) {
            case 1:
            case 2:
                System.out.println( "one, two, buckle my shoe" );
                break;
            case 3:
            case 4:
                System.out.println( "three, four, shut the door" );
                break;
            case 5:
            case 6:
                System.out.println( "five, six, pick up sticks" );
                break;
            case 7:
            case 8:
                System.out.println( "seven, eight, lay them straight" );
                break;
            case 9:
            case 10:
                System.out.println( "nine, ten, a big fat hen" );
                break;
        } // end switch
    } // end main()
} // end of class ex2c
```

branching with switch (4): using default.

```
public class ex2d {
    public static void main ( String[] args ) {
        int i = (Integer.valueOf( args[0] )).intValue();
        switch( i ) {
            case 1:
            case 2:
                System.out.println( "one, two, buckle my shoe" );
                break;
            case 3:
            case 4:
                System.out.println( "three, four, shut the door" );
                break;
            case 5: case 6:
                System.out.println( "five, six, pick up sticks" );
                break;
            case 7: case 8:
                System.out.println( "seven, eight, lay them straight" );
                break;
            case 9: case 10:
                System.out.println( "nine, ten, a big fat hen" );
                break;
            default:
                System.out.println( "nothing left to say!" );
                break;
        } // end switch
    } // end main()
} // end of class ex2d
```

cs3101-003-java-spring2003-sklar-lect02

5

looping (1).

- if you want to do something many times
- two modes of loops:
 - counter controlled (now)
 - condition controlled (later)
- three loop statements:
 - for
 - while
 - do
- you can actually do both modes with each of the three statements, though some mode/statement pairings are more common than others

cs3101-003-java-spring2003-sklar-lect02

6

looping (2): counter-controlled for.

```
public class ex2e {
    public static void main ( String[] args ) {
        int n, i;
        n = (Integer.valueOf( args[0] )).intValue();
        System.out.println( "counting up to " + n + "..." );
        for ( i=0; i<n; i++ ) {
            System.out.print( i+ " " );
        } // end for
        System.out.println();
    } // end of main
} // end of class ex2e
```

cs3101-003-java-spring2003-sklar-lect02

7

looping (3): counter-controlled while.

```
public class ex2f {
    public static void main ( String[] args ) {
        int n, i;
        n = (Integer.valueOf( args[0] )).intValue();
        System.out.println( "counting up to " + n + "..." );
        i = 0;
        while ( i < n ) {
            System.out.print( i+ " " );
            i++;
        } // end while
        System.out.println();
    } // end of main
} // end of class ex2f
```

cs3101-003-java-spring2003-sklar-lect02

8

looping (4): counter-controlled do.

```
public class ex2g {
    public static void main ( String[] args ) {
        int n, i;
        n = (Integer.valueOf( args[0] )).intValue();
        System.out.println( "counting up to " + n + "..." );
        i = 0;
        do {
            System.out.print( i+ " " );
            i++;
        } while ( i<n );
        System.out.println();
    } // end of main
} // end of class ex2g
```

looping (5): break and continue.

- these statements interrupt the normal flow of control of a program
- `break` is used in the `switch` statement to jump out of a case clause, without dropping down into the next one
- `break` can also be used from within a loop to interrupt the loop and jump to the end of the loop
- if loops are nested, it only jumps out of the loop where the `break` is imbedded
- `continue` is used from within a loop to interrupt the loop and jump to the next iteration of the loop
- in general, these statements are bad to use because they allow you to write code that jumps around and may be more prone to errors

looping (6): other facts about loops.

- you don't always have to count up
- you can count down too
- you don't always have to count by ones
- you can increment or decrement by any integer
- `do` loops always execute at least once
- `for` and `while` loops can be defined so that they don't execute (sometimes you might want to do this)

classes (1).

- *classes* are the block around which Java is organized
- classes are composed of
 - data elements
 - * *variables*
 - * *constants*
 - like variables, they have a type, a name and a value
 - *methods*
 - * modules that perform actions on the data elements
 - like variables, they have a type, a name and a value
 - unlike variables, the type can be *void*
 - * *constructors*
- classes are *hierarchical*
- groups of related classes are organized into *packages*
- we'll start looking at *native* packages

classes (2): the `java.lang` package.

- the superclass for all Java classes, at the top of the hierarchy
 - `java.lang.Object`
- *wrapper* classes that wrap around primitive data types; classes that define numeric limits and contain conversion methods
 - `java.lang.Boolean`
 - `java.lang.Character`
 - `java.lang.Byte`, `java.lang.Short`, `java.lang.Integer`,
`java.lang.Long`, `java.lang.Float`, `java.lang.Double`
- string handling functions
 - `java.lang.String`
- math functions
 - `java.lang.Math`

classes (3): `java.lang.Integer` class.

- a *constructor*:
`public Integer( int value );`
- some *constants*:
`public static final int MIN_VALUE`
`public static final int MAX_VALUE`
- some *methods*:
`public int intValue();`
`public static String toString( int i );`
`public static Integer valueOf( String s );`
`public static int parseInt( String s );`
- there is one for each primitive data type
- exercise:
use the on-line Java documentation to look up the name of the wrapper classes for each of the primitive data types

classes (4): `java.lang.String` class.

- some *constructors*:
`public String();`
`public String( String value );`
- some *methods*:
`public static String valueOf( int i );`
`public int charAt( int index );`
`public int compareTo( String anotherString );`
`public int length();`

classes (5): `java.lang.Math` class.

- some *constants*:
`public static final double E`
`public static final double PI`
- some *methods*:
`public static int abs( int a );`

`public static native double sin( double a );`
`public static native double cos( double a );`
`public static native double tan( double a );`

`public static native double pow( double a, double b );`
`public static native double sqrt( double a );`

`public static double random();`

classes (6): java.util.Random class (1).

- there is another way to generate random numbers besides using the `Math.random()` from the `java.lang.Math` class

- there are two methods defined in the `Random` class:

```
public Random();  
public Random( long seed );  
// constructor -- can be called with or without a seed
```

```
public void setSeed( long seed );  
// sets the seed for the random number generator
```

- this class implements a *pseudo random number generator*
- which is really a sequence of numbers
- the *seed* tells the random number generator where to start the sequence

classes (7): java.util.Random class (2).

- more methods defined in the `Random` class, used to get the random numbers:

```
public float nextFloat();  
// returns a random number between 0.0 (inclusive) and  
// 1.0 (exclusive)
```

```
public int nextInt();  
// returns a random number that ranges over all possible  
// int values (positive and negative)
```

classes (8): java.util.Date class (1).

- this class is handy for getting the current date
- or creating a `Date` object set to a certain date
- some methods defined in the `Date` class:

```
public Date();  
public Date( long date );  
// constructor -- called without an argument, uses the  
// current time; otherwise uses the time argument
```

```
public boolean after( Date arg );  
public boolean before( Date arg );  
public boolean equals( Object arg );  
public long getTime();  
public String toString();
```

- computer time is measured in milliseconds since midnight, January 1, 1970 GMT
- a `Date` object is handy to use as a seed for a random number generator

classes(9): instantiating objects.

- in order to use a class, you *instantiate* it by creating an *object* of that type
- this is kind of like declaring a variable

```
import java.util.*;  
public class ex2h {  
    public static void main( String[] args ) {  
        Date now = new Date();  
        Random rnd = new Random( now.getTime() );  
        System.out.println( "here's the first random number: "+  
                             rnd.nextInt() );  
    } // end of main()  
} // end of class ex2h
```

more looping (1).

- back to loops
- condition-controlled loops

more looping (2): condition-controlled while.

```
public class ex2i {
    public static void main ( String[] args ) {
        int card1=(int)(Math.random()*52);
        int card2=(int)(Math.random()*52);
        int count=1;
        while ( card1 != card2 ) {
            System.out.println( "count="+count+" card1="+card1+
                               " card2="+card2 );
            card1=(int)(Math.random()*52);
            card2=(int)(Math.random()*52);
            count++;
        } // end while
        System.out.println( "MATCH! count="+count+" card1="+card1+
                           " card2="+card2 );
        System.exit( 0 );
    } // end of main
} // end of class ex2i
```

more looping (3): condition-controlled do.

```
public class ex2j {
    public static void main ( String[] args ) {
        int card1=(int)( Math.random()*52 );
        int card2=(int)( Math.random()*52 );
        int count=1;
        do {
            System.out.println( "count="+count+" card1="+card1+
                               " card2="+card2 );
            card1=(int)( Math.random()*52 );
            card2=(int)( Math.random()*52 );
            count++;
        } while ( card1 != card2 );
        System.out.println( "MATCH! count="+count+" card1="+card1+
                           " card2="+card2 );
        System.exit( 0 );
    } // end of main
} // end of class ex2j
```

more looping (3): condition-controlled for.

```
public class ex2k {
    public static void main ( String[] args ) {
        int card1=(int)( Math.random()*52 );
        int card2=(int)( Math.random()*52 );
        int count=1;
        for ( ; card1 != card2; ) {
            System.out.println( "count="+count+" card1="+card1+
                               " card2="+card2 );
            card1=(int)( Math.random()*52 );
            card2=(int)( Math.random()*52 );
            count++;
        } // end for
        System.out.println( "MATCH! count="+count+" card1="+card1+
                           " card2="+card2 );
        System.exit( 0 );
    } // end of main
} // end of class ex2k

OR you can include all updates in the update section of the for loop:
for ( ; card1 != card2; card1=(int)(Math.random()*52),
    card2=(int)(Math.random()*52),
    count++ ) {
    System.out.println( "count="+count+" card1="+card1+
                       " card2="+card2 );
} // end for
```

inheritance (1).

- *inheritance* is the means by which classes are created out of other classes
- it is a cornerstone of object-oriented programming
- the idea is to create classes that can be re-used from one application to another
- classes contain *data objects* and *methods*
- you want to be able to change the *data type* of the data objects and still be able to use the same methods
- you also want to be able to change the flavor of what the methods do

inheritance (2).

- think of the most primitive Java class, `Object` as being at the root of the inheritance tree
- all other classes are “children” or *subclasses* of that class
- here is an example of the inheritance tree for `Integer`:

```
java.lang.Object
|
+-- java.lang.Number
|
+-- java.lang.Integer
```

- `Integer` is a subclass of `Number` and `Number` is a subclass of `Object`
- `Integer` is also a subclass of `Object`
- conversely a parent is also called a *superclass*
- `Object` is a superclass of `Number` and `Number` is a superclass of `Integer`
- `Object` is also a superclass of `Integer`
- `Object` is also called the *base class* of `Integer`

inheritance (3).

- as you move DOWN the inheritance tree from the root to the leaf, you are *extending* subclasses from parent classes
 - parent classes are also called *superclasses*
 - or *base classes*
 - children classes are *derived* from their parents
- as you move UP the inheritance tree from the leaf to the root, you can say that each subclass is a *more specific* version of its parent
- this is known as the *is-a* relationship between a subclass and the parent class that the child extends
- the keyword `this` is used to specify a member of the current or immediate class

putting it all together.

- exercise:
use the `String.charAt()` method and what we learned about command-line input, condition-controlled loops and `switch` statements to read a word from the user on the command-line and strip all leading consonants off the user's input and then print out what is left