cisc3120
design and implementation of software applications I
spring 2015
lecture # I.2

topics:

- introduction to java, part 2
  - branching with `switch`
  - looping
  - arrays
  - java classes
  - writing your own classes
  - other terminology

on-line resources:

- java API specification: `http://docs.oracle.com/javase/8/docs/`
- java tutorial “getting started”:
  `http://docs.oracle.com/javase/tutorial/getStarted/`

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");
    }
  }
}
```

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;
    }
  }
}
```

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;
    }
  }
}
```

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;
    }
  }
}
```

looping (1).

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

looping (2): counter-controlled `for`.

```
public class Ex2e {
  public static void main(String[] args) {
    int n = (Integer.valueOf(args[0])).intValue();
    System.out.println("counting up to " + n + "...");

    for(int i = 0; i < n; i++) {
      System.out.print(i + "  ");
    }
    System.out.println();
  }
}
```

looping (3): counter-controlled `while`.

```
public class Ex2f {
  public static void main(String[] args) {
    int n = (Integer.valueOf(args[0])).intValue();
    System.out.println("counting up to " + n + "...");

    int i = 0;
    while(i < n) {
      System.out.print(i + "  ");
      i++;
    }
    System.out.println();
  }
}
```

looping (4): counter-controlled `do`.

```
public class Ex2g {
  public static void main(String[] args) {
    int n = (Integer.valueOf(args[0])).intValue();
    System.out.println("counting up to " + n + "...");

    int i = 0;
    do {
      System.out.print(i + "  ");
      i++;
    } while(i < n);
    System.out.println();
  }
}
```

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

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)

more looping (2): condition-controlled `while`.

```
public class Ex2h {
  public static void main(String[] args) {
    final int numCards = 52;
    int card1 = (int)(Math.random() * numCards);
    int card2 = (int)(Math.random() * numCards);
    int count = 1;
    while(card1 != card2) {
      System.out.println("count=" + count + " card1=" + card1 +
                         "card2=" + card2);
      card1 = (int)(Math.random() * numCards);
      card2 = (int)(Math.random() * numCards);
      count++;
    }
    System.out.println("MATCH! count=" + count + " card1=" + card1 +
                       " card2=" + card2);
  }
}
```

more looping (3): condition-controlled `do`.

```
public class Ex2i {
  public static void main(String[] args) {
    final int numCards = 52;
    int card1, card2;
    int count = 1;
    do {
      card1 = (int)(Math.random() * numCards);
      card2 = (int)(Math.random() * numCards);
      System.out.println("count=" + count + " card1=" + card1 +
                         " card2=" + card2);
      count++;
    } while(card1 != card2);
    System.out.println("MATCH! count=" + count + " card1=" + card1 +
                       " card2=" + card2);
  }
}
```

more looping (3): condition-controlled `for`.

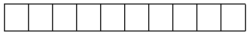
```
public class Ex2j {
  public static void main(String[] args) {
    final int numCards = 52;
    int card1 = (int)(Math.random() * numCards);
    int card2 = (int)(Math.random() * numCards);
    int count = 1;
    for(; card1 != card2;) {
      System.out.println("count=" + count + " card1=" + card1 +
                         " card2=" + card2);
      card1 = (int)(Math.random() * numCards);
      card2 = (int)(Math.random() * numCards);
      count++;
    }
    System.out.println("MATCH! count=" + count + " card1=" + card1 +
                       " card2=" + card2);
  }
}
```

OR you can include all updates in the update section of the for loop:

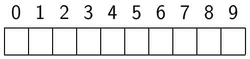
```
for (; card1 != card2; card1 = (int)(Math.random() * numCards),
                       card2 = (int)(Math.random() * numCards),
                       count++) {
  System.out.println("count=" + count + " card1=" + card1 +
                     " card2=" + card2);
}
```

arrays (1).

- used to associate multiple instances of the same type of variable
- the "`[]`" indicates it's an *array*
- we can have arrays of anything (i.e., other data types)
- one example we've already used is `String[]`, which is an array of `String`...
- visualize an array as a sequence of boxes, contiguous in the computer's memory, where each box stores one instance of the type of data associated with that array:



- the boxes are numbered, starting with 0 and ending with the length of the array less one; each number is called an *index*
- the *indices* for an array of 10 items can be visualized like this:



arrays (2).

- to use an array, first you must declare it:

  `int[] A;`

- then you must instantiate it:

  `A = new int[10];`

- or you can do both of these in one step:

  `int[] A = new int[10];`

- then you can access its elements:

  `A[4]`

  (index=4, which is the 5th item in the array...)

- you can use this accessed item just like any single data element of that type, in this case an `int`
- the number of items in the array is the variable `A.length`

arrays (3).

- here's an example that stores in an array 5 random numbers between 0 and 100:

```
public class Ex2k {
  public static void main(String[] args) {
    int[] A = new int[5];
    for(int i = 0; i < A.length; i++) {
      A[i] = (int)(Math.random() * 100);
    }
    for(int i = 0; i < A.length; i++) {
      System.out.println("i[" + i + "]=" + A[i]);
    } // end for i
  } // end of main()
} // end of class Ex2k
```

two-dimensional arrays.

- arrays of arrays
- also called a two-dimensional array
- two-dimensional arrays are declared like this:
  `char[][] a2;`
- and instantiated like this (for example for a 5x5 array):
  `a2 = new char[5][5];`
- the first dimension is called *row*
- the second dimension is called *column*
- so the element in the i-th row and the j-th column is accessed like this:
  a2[i][j]

java classes (1).

- *classes* are the block around which Java is organized
- classes are composed of
  - data elements
    - *variables* — i.e., their values can change during the execution of a program
    - *constants* — i.e., their values CANNOT change during the execution of a program
      - 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* — special types of methods used to set up an object before it is used for the first time
- classes are *hierarchical*
- groups of related classes are organized into *packages*
- we'll start looking at *native* packages

java 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`

java 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

java 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();
```

java 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();
```

java 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

java 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)
```

java 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

java 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 Ex21 {
  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 Ex21
```