Today’s Topics

• Comments and/or Questions?
• Finish discussion of this
• ternary conditional operator
• Review of ways to call methods
• DecimalFormat class
• Overloading methods
• Object references and passing primitives vs. passing object references / array references
• Is-a vs. Has-a relationships
• this
  – is a way to explicitly say you are referring to an instance variable inside the same class
  – it contains the value of the “current object”
  – I'll show an example of this in the Card class.
• example usage: this.rank (would refer to the rank instance variable within a method of the Card class)
• Why use this?
  – if you name a parameter of a method the same name as an instance variable, you can
    • refer to the instance variable inside the method by using this. in front of it
    • and refer to the parameter by its name alone
Ternary operator (acts like if / else)

- Ternary conditional operator contains three parts separated by a ? and a :

Example:
int hour = 14, nonMilHr;
nonMilHr = ((hour == 12 || hour == 0) ? 12 : hour % 12 );
ways to call methods

• a method that is defined in the same class as another method can call it by just using the method name and arguments in parentheses (regardless of whether the method is public or private).

• we saw many examples of this: one was in the sorting program we wrote --- the swap method was defined in our class and was called like so

```java
swap(arr, i, i+1);
```
ways to call methods

• a public static method that is defined in some other class can be called by using the class name followed by the dot operator and then the method name and arguments in parentheses. Note: we don’t have to first define a reference to an object of that class.

• we’ve seen examples of this with the Integer.parseInt and JOptionPane.showMessageDialog methods (among many others)

• We do NOT need to declare a variable to hold a reference to an object of these classes if all we want to do is call a static method in it. We call one of these like so:

```java
String someStr = "34";
int someInt;
someInt = Integer.parseInt(someStr);
```
ways to call methods

• A non-static public method that is defined in some other class must be called by using a reference to the object of the class followed by the dot operator and then the method name and arguments in parentheses.

• we’ve seen examples of this with the Card class

• First we declare a variable to hold a reference to a Card object, then we can call one of Card’s methods.

• E.g. the call to toString() in the second line below

  Card myCard = new Card(2,2);
  String s = myCard.toString();
  System.out.print(" the card is " + s);
ways to call methods

• A private method that is defined in some other class cannot be called from outside that class.
Recap - ways to call methods

- a method (regardless of whether the method is public or private) that is defined in the same class as another method can call it by just using the method name and arguments in parentheses.
  
  ```java
  swap(arr, i, i+1);
  ```

- a non-static public method that is defined in some other class (from where it is being called) must be called by using a reference to the object of the class followed by the dot operator and then the method name and arguments in parentheses.

  ```java
  Card myCard = new Card(2,2);
  String s = myCard.toString();
  System.out.print(" the card is " + s);
  ```

- a public static method that is defined in some other class can be called by using the class name followed by the dot operator and then the method name and arguments in parentheses.

  ```java
  some_int = Integer.parseInt(some_str);
  ```
DecimalFormat class

- Let's look online at the Java API for this class
- Allows us to make output nicer by rounding floats and doubles etc. for output
- Specify a pattern when creating an object of DecimalFormat class
- 0 vs. #
  - 0 is a required digit position
    - To the left of decimal --- means at least that many
    - To the right of decimal --- means exactly that many places
  - # to the right of decimal means that a 0 will not display.
- % at end of pattern causes number to be displayed as a percent
- E-notation is also possible (scientific notation)
DecimalFormat class

• DecimalFormat df = new DecimalFormat("00.00");

• To use it: (assuming we have a double var temperature)
  – df.format(temperature)
    /* this will return a String based on the value of
    temperature and the pattern specified when the df
    object was constructed */
  – Since it returns a String we can place it anywhere a
    String can be used, like:
    System.out.println("The temperature is: " + df.format(temperature));
DecimalFormat class

• Let's try a few different formats and see what is produced.
• Note: no truncation of numbers to the LEFT of the decimal point will ever occur with this formatting even if we have:

  – the format is 00.0#

  – assume the value to format is 365.2512

  – The number will be formatted as 365.25 because the amount of 0's to the LEFT of the decimal in the format means at least that many places.
overloading methods

• recall that we overloaded the constructor --- what did that mean?
• we can do the same with any method.
overloading methods

• recall that we overloaded the constructor --- what did that mean?
• we can do the same with any method.
  – we give a method the same name but different number of parameters and/or types of parameters. Order matters. names of parameters do not. A method's signature must be unique. Signature includes name of method, and types of parameters in order.
  – e.g.
    • public void meth1(int i, int j)
    • public void meth1(int j)
    • public void meth1(String s, int i, int j)
    • public void meth1(int i, String s, int j)
    • public void meth1(int j, int k)
      – // this last one not allowed b/c signature is same as first one.
Object References

• When we declare an object to be of some Class type, we are declaring a REFERENCE to an object of that class.

• The reference stores the “memory address” of where the actual data for the object is stored.

• e.g
  – String s = “Mike”;
  – // s is the name of the reference to a String.
  – // The value in s is a memory address
  – // that memory address is where to find the contents of the string: “Mike”
Object References

• When we assign one object reference to another
  – the references both contain the same memory address
• A main implication is that changing one reference's object's values will affect the other.

Card c1 = new Card(1, “Clubs”);  // Ace of Clubs
Card c2;
c2 = c1;
c1.changeCard(2, “Diamonds”);  // 2 of Diamonds
// since c1 and c2 both reference the SAME memory
// address, changing one's values changes the others
// because c1 and c2 are different names for the same memory
Object References

- All variables passed into methods have a local copy of them made each time the method executes.
  - Primitive type variables
    - When variables of primitive types are passed into a method the value that is copied is the actual value of the data in the variable
  - Object References (and array references)
    - When we pass an object reference to a method the value that is copied is the memory address, NOT the contents at that memory address.
Passing primitives

• Example with primitives:

```java
public static void meth1(int parm)
{
    parm++;  
    System.out.println("value = " + parm);
}
```

// Then in main method a call to it might be like:
int x = 5;
meth1(x);

// parm has the value 5 copied into it when meth1 is called
// meth1 works on that copy, so x's value isn't altered in meth1
Passing References

• Example with arrays:

```java
class Test {
    public static void meth2(int parm[]) {
        parm[0] = 4;
        System.out.println("value = " + parm[0]);
    }
    static {
        int x[] = { 3, 1, 2, 5 };
        System.out.println("Before call:");
        System.out.println(Arrays.toString(x));
    }
    public static void main(String[] args) {
        meth2(x);
        System.out.println("After call:");
        System.out.println(Arrays.toString(x));
    }
}
```

// Then in main method a call to it might be like:
int x[] = { 3, 1, 2, 5 };
meth2(x);

// Array names like x here are references, parm gets a copy of
// the memory address of x. meth2 works on that address, so x's
// array values are altered in meth2
Passing References

• Example with object references:

```java
public static void meth3(Card parm)
{
    parm.changeCard(11, "Clubs");  // Jack of Clubs
}
```

// Then in main method a call to it might be like:
Card acard = new Card(1, "Clubs");  // Ace of Clubs
meth3(acard);

// Object names like acard are references, parm gets a copy of
// the memory address of acard.  meth1 works on that address, so
// acard's object values are altered in meth3
References vs. Primitive variables

Card c = new Card(2,2);
Card d;
d = c;
d.setRank(12);
System.out.println(c.toString());
System.out.println(d.toString()); // c and d refer to the same data

int x = 2;
int y;
y = x;
y = 12;
System.out.println("x = " + x);
System.out.println("y = " + y); // x and y refer to different data
Is-a vs. Has-a

• A “has a” relationship is one that corresponds to a class and its data members.

• For example:
  – A Card has a suit and has a rank
  – A Deck has 52 Cards
  – A Car has a color and has a make and has a model, ...
  – An Employee has a hire date and has a salary, ...
  – Can we think of any more?
Is-a vs. Has-a

• A “has a” relationship is one that corresponds to a class and its data members.

```java
public class Card {
    private int suit;
    private int rank;
    // ...
}
```

```java
public class Car {
    private String color;
    private String make;
    // ...
}
```
Is-a vs. Has-a

• A “has a” relationship is one that corresponds to a class and its data members.

```java
public class Date {
    private int year, month, day;
    // ...
}
```

```java
public class Employee {
    private Date hire_date;
    private double salary;
    // ...
}
```
Is-a vs. Has-a

• An “is a” relationship is one that corresponds to a class and the class from which it inherits. Also can be said as “is a type of”.

• For example:
  – A Faculty member is an Employee
  – A Staff member is an Employee
  – A Car is a Vehicle
  – A Truck is a Vehicle
  – A Motorcycle is a Vehicle

  – Can we think of any more?
Other inheritance relationships

• Can you think of anything else that has a “has a” relationship?
  − Rooms have doors, windows, etc.

• Can you think of anything else that has an “is a” relationship?
  − Bedroom is a Room
  − Dining room is a Room

  − How about Animals?