Reading:
1. Deitel & Deitel, Chapters 10, 14

Objectives:
1. Continue discussing Object Oriented Programming
2. Learn more about Files and Streams

Concepts:
1. Polymorphism
2. Files & Streams

References:
2. http://java.sun.com/j2se/1.4.2/docs/api/
1. Polymorphism:

Classes in a hierarchy are related by the "is-a" relationship. For example, a Nissan is-a Automobile, and a Sentra is-a Nissan. This chapter discusses how reference variables are used for objects from different classes within a hierarchy. Another subject is the idea of an abstract class — a class that cannot be instantiated but that can be the parent of other classes.

- Abstract Classes.
- Abstract Methods.
- Polymorphism.

Example - Greeting Card Hierarchy

The example of this chapter is a hierarchy of greeting cards types. The parent class is Card, and its children classes are Valentine, Holiday, and Birthday.

A card object will have a greeting() method that writes out a greeting. Each type of card contains an appropriate greeting. For example, the birthday card will have a greeting saying “Happy Birthday”

In this example, an object must be an instance of one of the three child types: Valentine, Holiday, and Birthday. There will be no such thing as an object that is merely a "Card." The Card class represents the abstract concept of "Card." All actual card objects must be more specific.
Abstract Class:
For this example, all card objects are one of the three types (Valentine, Holiday, or Birthday) and the parent class Card is used only to group them into a hierarchy. There will never be an object that is just a card. This is useful to do, just as a store has all its various greeting cards in one display, arranged into several categories.

An **abstract class** in Java is a class that is never instantiated. Its purpose is to be a parent to several related classes. The child classes inherit from the abstract parent class.

In hierarchy drawings (such as on the previous page), abstract classes are drawn with dotted lines.

Access modifiers such as `public` can be placed before `abstract`. Even though it can not be instantiated, an abstract class can define methods and variables that children classes inherit.

Abstract classes are a way of organizing a program. You can get the same thing done without using this way to organize. This is a matter of program design, which is not easy at all.

Abstract Methods:

In this example, each card class has its own version of the `greeting()` method. Each class has a `greeting()`, but each one is implemented differently. It is useful to put an abstract `greeting()` method in the parent class. This says that each child inherits the "idea" of `greeting()`, but each implementation is different. Here is the class definition of the abstract class `Card`:

```java
abstract class Card {
  String recipient; // name of who gets the card
  public abstract void greeting(); // abstract greeting() method
}
```

This is the complete definition of this abstract class. Notice the **abstract method**. Abstract classes can (but don't have to) contain abstract methods. Also, an abstract class can contain non-abstract methods, which will be inherited by the children.

Polymorphism:

**Polymorphism** means "having many forms." In Java, it means that a single variable might be used with several objects of related classes at different times in a program. When the variable is used with "dot notation" variable.method() to invoke a method,
exactly which method is run depends on the object that the variable currently refers to. Here is an example:

. . . .                           // class definitions as before

public class CardTester
{
    public static void main ( String[] args ) throws IOException
    {

        Card card = new Holiday( "Amy" );
        card.greeting();                      //Invoke a Holiday greeting()

        card = new Valentine( "Bob", 3 );
        card.greeting();                      //Invoke a Valentine greeting()

        card = new Birthday( "Cindy", 17 );
        card.greeting();                      //Invoke a Birthday greeting()

    }
}

2. Files & Streams
   17.1 Introduction
   17.2 Data Hierarchy
   17.3 Files and Streams
Create Holiday Cards with Greetings using the concept of Abstract Classes

Solution:

```java
import java.io.*;

abstract class Card {
    String recipient;
    public abstract void greeting();
}

class Holiday extends Card {
    public Holiday(String r) {
        recipient = r;
    }
    public void greeting() {
        System.out.println("Dear " + recipient + ",");
        System.out.println("Season's Greetings!\n\n");
    }
}

class Birthday extends Card {
    int age;
    public Birthday(String r, int years) {
        recipient = r;
        age = years;
    }
    public void greeting() {
        System.out.println("Dear " + recipient + ",");
        System.out.println("Happy " + age + "th Birthday\n\n");
    }
}
```
class Valentine extends Card
{
    int kisses;

    public Valentine ( String r, int k )
    {
        recipient = r;
        kisses    = k;
    }

    public void greeting()
    {
        System.out.println("Dear " + recipient + ", \\
        Love and Kisses, \\
        for ( int j=0; j < kisses; j++ )
            System.out.print("X");
        System.out.println("\n\n");
    }
}

public class CardTester
{
    public static void main ( String[] args ) throws IOException
    {
        String me;
        BufferedReader input = new BufferedReader( new InputStreamReader(System.in) );
        System.out.println("Your name:");
        me = input.readLine();

        Holiday   hol = new Holiday( me );
        hol.greeting();

        Birthday  bd  = new Birthday( me, 21 );
        bd.greeting();

        Valentine val = new Valentine( me, 7 );
        val.greeting();
    }
}