Abstract Data Types (ADT)

An **abstract data type** (ADT) is a set of data values and associated operations that are precisely specified independent of any particular implementation. The String class, for example is an abstract data type provided by the Java programming language. The String class is comprised of an array of characters, (the data values) and provides an assortment of methods such as length and charAt which performs operations on the String object (associated operations).

Typically, when defining an ADT, the data values are given private access while the methods that are intended to be available to the user are given public access. Assigning private access to the data is an example of information hiding. The user does not need to know about the underlying structure of the ADT, he simply needs to know how to use it, or call its methods.

Since the data values are normally given private access you generally implement methods that allow the user to access the data values (accessors). And, depending on the purpose of the ADT, you may also provide the user with methods that allow the data values to be changed (mutators).

For example, the following class stores a person's first and last names. It provides appropriate constructors, accessors, and mutators, a toString method that would be displayed if the an object of the Person class was passed as argument in a print or println statement, and an equals method that allows this object to be compared with another object (of the same class).

```
public class Person
{
    private String firstName;
    private String lastName;

    // Empty constructor
    Person() { }                              Person p = new Person("Bill", "Smith");

    // Constructor that receives a persons first and last name
    Person(String first, String last) {
        firstName = first;
        lastName = last;                       firstName = "Bill"
    }                                          lastName = "Smith"

    // Accessor for firstName field
    public String getFirstName() {
        return firstName;
    }
    // Accessor for lastName field
    public String getLastName() {
        return lastName;
    }

    // Mutator for firstName field
    public void setFirstName(String first) {
        firstName = first;
    }
```

If a **Person** object is created using the default constructor then both instance fields must be initialized before the Object can be used.

```
    Person p = new Person( );
    p.setFirstName("Bill");
    p.setLastName(“Smith”);
```

```
    // Mutator for lastName field
    public void setLastName(String name) {
        lastName = name;
    }

    // This method returns a String representation of this object
    public String toString() {
        return “[“ + lastName + “, “ + firstName + “]”;
    }

    // This method returns true if the first and last names of this
    // argument are the same
    public boolean equals(Object obj) {
        if (obj instanceof Person)
        {
            Person p = (Person) obj;
            return lastName.equals(p.lastName) && firstName.equals(p.firstName);
        }
        return false;
    }
}
```

Test **obj** to determine if it IS an instance of *this* class (**Person**).

- Typecast **obj** into an object of *this* class (**Person**) if it IS an instance of *this* class (**Person**).
- Otherwise, return **false**.

The Person class provides a convenient way to store a person’s first and last name without the necessity of creating parallel arrays. A single array of type Person is all that is required. The following program creates an array of 100 Persons, reads in a list of names from a data file, and prints the names by invoking the toString method of the Person class.

```
import java.util.*;
import java.io.FileReader;
import java.io.IOException;

public class MyFriends
{

    private Person[ ] people = new Person[15];   // an array of 15 Person objects
    private int count;                           // the number of elements stored in people
    public static void main(String args[]) {
        new MyFriends();
}

    MyFriends() {
        input();
        output();
    }

    private void input() {
        try
        {
            Scanner reader = new Scanner(new File("friends.dat"));

            count = 0;
            while(reader.hasNextLine())
            {
                String first = reader.next();          // read first name from keyboard
                String last = reader.next();           // read last name from keyboard
                Person p = new Person(last, first);    // create a person object
                people[count++] = p;                   // add the person object to the array
            }
        }
        catch (FileNotFoundException e)
        {
            System.out.println("Error reading friends.dat!");
            System.exit(0);
        }
    }

    // We pass the Person object p to the println() method thereby invoking the toString()
    // method of the Person class.
    private void output() {
        for (int i = 0; i < count; i++) // process every Person in the array
            System.out.println(people[i]);   // print the Person p
    }
}
```

If the **"friends.dat"** file contained the following list of names:

```
Bob Hope
Norman Rockwell
Clint Eastwood
Roy Rogers
Lana Turner
Steve Reeves
```

```
James Hickcock
Martin Short
Clark Gable
Red Skelton
Howard Johnson
Suzanne Summers
Dale Evans
Gene Autry
Wyatt Earp
```

Then running MyFriends results in the following output:

```
[Hope, Bob]
[Rockwell, Norman]
[Eastwood, Clint]
[Rogers, Roy]
[Turner, Lana]
[Reeves, Steve]
[Hickcock, James]
[Short, Martin]
[Gable, Clark]
[Skelton, Red]
[Johnson, Howard]
[Summers, Suzanne]
[Evans, Dale]
[Autry, Gene]
[Earp, Wyatt]
```

Lab 18 - ASSIGNMENT

Lab 18A - 70 points

OBJECTIVE

WAC (write a class) that stores the name of a city and the state/country in which it resides. Call the class `City`. `City` is an *Abstract Data Type* and WILL NOT contain a main method.

FIELD SUMMARY

- **String city** – the city name
- **String country** – the city location (country or U. S. state.

METHOD SUMMARY

- **constructors** – create an empty constructor and a constructor that receives the name of a city and a country (or U.S. State) as arguments and assigns those values to the corresponding instance variables.
- **accessors** & **mutators** – one for each instance variable.
- **equals** – compares two objects to determine if they are the same. (They are the same if the *city* and *country* are the same).
- **toString -** returns a String representing the City object. The String returned by the `toString` method should have the following appearance: **city, country**

OBJECTIVE

WAP called `FamousPlaces`. `FamousPlaces` will read in the names of cities and countries from two data files ("city1.dat" and "city2.dat"). Each file contains exactly the same number of cities and countries. Store the cities and countries in two parallel arrays of `City` objects (`city1` and `city2`). Compare the contents of the two arrays, element by element, and output whether the two cities are the same.

FIELD SUMMARY

- **City[ ] city1 –** An array of City objects
- **City[ ] city2 –** A parallel array of City objects
- **int count –** an accumulator to keep track of the number of cities in both arrays.

METHOD SUMMARY

- **main** – create an instance of this class. Make method calls to `input` and `output`.
- **input** – Declare a `Scanner` object and read all the cities and countries from the data file ("city1.dat") into the array `city1`. Declare a new `Scanner` object and read all the cities and countries from the data file ("city2.dat") into the array `city2`.
- **output** – Compare the names of each `City` object in the two arrays and display the results on the console screen.

SAMPLE DATA FILE INPUT

“city1.dat”	**“city2.dat”**
PARIS FRANCE	PARIS TEXAS
ATHENS GREECE	ATHENS GEORGIA
LONDON ENGLAND	LONDON ENGLAND
TOKYO JAPAN	OSAKA JAPAN
SEOUL KOREA	SEOUL KOREA
ROME ITALY	FLORENCE ITALY
MADIRD SPAIN	MADRID SPAIN
CAIRO EGYPT	ALEXANDRIA EGYPT
MOSCOW RUSSIA	MOSCOW RUSSIA
OSLO MINNESOTA	OSLO NORWAY

SAMPLE OUTPUT

```
PARIS, FRANCE is NOT the same as PARIS, TEXAS
ATHENS, GREECE is NOT the same as ATHENS, GEORGIA
LONDON, ENGLAND is the same as LONDON, ENGLAND
TOKYO, JAPAN is NOT the same as OSAKA, JAPAN
SEOUL, KOREA is the same as SEOUL, KOREA
ROME, ITALY is NOT the same as FLORENCE, ITALY
MADIRD, SPAIN is NOT the same as MADRID, SPAIN
CAIRO, EGYPT is NOT the same as ALEXANDRIA, EGYPT
MOSCOW, RUSSIA is the same as MOSCOW, RUSSIA
OSLO, MINNESOTA is NOT the same as OSLO, NORWAY
```

Lab 18B - 80 points

OBJECTIVE

WAC that stores names, address, etc. for individual persons (an ADT). Call the class `Address`. `Address` is an *Abstract Data Type* and WILL NOT contain a main method.

FIELD SUMMARY

- **String firstName**
- **String lastName**
- **String address**
- **String city**
- **String state**
- **String zipCode**
- **String phoneNumber**

METHOD SUMMARY

- **constructors** – create an empty constructor and a constructor that receives an argument corresponding to each instance variable and assigns those values to the corresponding instance variables.
- **accessors** & **mutators** – one for each instance variable.
- **equals** – compares two objects to determine if they are the same. (They are the same if the `firstName`, `lastName`, and `address` are the same).
- **toString** - returns a String representing the `Address` object. The String returned by the `toString` method should have the following appearance:
  [firstName lastName/address, city, state, ZipCode/telephone number]

OBJECTIVE

WAP called `LittleBlackBook`. `LittleBlackBook` is an address book that holds the names, addresses, telephone numbers, etc. of friends and acquaintances. `LittleBlackBook` will implement an array of `Addresses` to hold the various names and addresses.

FIELD SUMMARY

- **Address[ ] addresses –** An array of `Address` objects
- **int count –** an accumulator to keep track of the number of addresses in the arrays.

METHOD SUMMARY

- **main** – create an instance of this class. Make a method call to input and output.
- **input** – declare a `Scanner` object that reads from a data file ("lab21b.dat") and read the names, addresses, etc. adding them to the array as new `Address` objects. Using a regular expression can greatly simplify the input process. Since each piece of data is separated by a comma but each line ends with a RETURN we need to be able to delimit by either a comma

or a RETURN. The regular expression "**,|\r\n**" will allow us to do just that. The expression means to delimit on a comma (**,**) or (**|**) a RETURN (**\r\n**).

- **output** –display all the names and addresses on the console screen.

SAMPLE DATA FILE INPUT

```
Gobel,George,1418 Short Street,Los Angeles,Ca,47123,214-932-3847
Anthony,Susan,1234 Post Oak Blvd,Houston,Tx,77314,832-281-1453
Kirk,James,111 New Worlds Lane,South Trek,Az,38472,615-382-2938
Cassidy,Hopalong,342 Pony Express Street,Helena,Montana,48574,892-394-4958
```

SAMPLE OUTPUT

```
[George Gobel/1418 Short Street, Los Angeles, Ca 47123/214-932-3847]
[Susan Anthony/1234 Post Oak Blvd, Houston, Tx 77314/832-281-1453]
[James Kirk/111 New Worlds Lane, South Trek, Az 38472/615-382-2938]
[Hopalong Cassidy/342 Pony Express Street, Helena, Montana 48574/892-394-4958]
```

Lab 18C - 90 points

OBJECTIVE

WAC that stores information pertaining to checks, i.e. number, date, payee, and amount. Call this class `Check`. The `Check` class will be used to store information about checks written on your bank account. This class is and ADT and DOES NOT contain a main method.

FIELD SUMMARY

- **int number** – check number.
- **String date** – date the check was written.
- **String payee** – who the check is made out to.
- **double amount** – amount of the check.

METHOD SUMMARY

- **constructors** – create an empty constructor and a constructor that receives an argument corresponding to each instance variable and assigns those values to the corresponding instance variables.
- **accessors** & **mutators** – one for each instance variable.
- **equals** – compares two objects to determine if they are the same. (They are the same if the `numbers` are the same).
- **toString -** returns a String representing the `Address` object. The String returned by the toString method should have the following appearance:
  [Check Number, Date, Payee, Amount]

OBJECTIVE

WAP called `CheckingAccount` that implements an array of `Check` objects. `CheckingAccount` will keep track of checks written on this account and provide the user with the starting and ending balances after processing any checks that were written.

FIELD SUMMARY

- **Check[ ] checks –** An array of `Check` objects
- **int count –** an accumulator to keep track of the number of checks in the arrays.
- **double startingBalance –** bank account starting balance.
- **double endingBalance –** bank account ending balance.

METHOD SUMMARY

- **main** – create an instance of this class. Make a method call to input, process, and output.
- **input** – declare a `Scanner` object that reads from a data file ("lab21c.dat") and read the starting balance followed by individual checks (number, date, payee, and amount) adding them to the array as new `Check` objects.
- **process** – deduct the amount of each check from the beginning balance resulting in an

ending balance. (Ending balance = starting balance – each check)

- **output** – display the starting balance, all the checks that were processed (using the Check `toString` method) and the ending balance.

SAMPLE DATA FILE INPUT

```
1103.21                 // The beginning balance
101
01/12/2004
Walmart
72.5
102
02/23/2004
Payless Shoes
49.95
103
03/12/2004
H.E.B.
26.5
```

SAMPLE OUTPUT

```
Beginning Balance: 1103.21

Checks Processed:
[101, 01/12/2004, Walmart, $72.50]
[102, 02/23/2004, Payless Shoes, $49.95]
[103, 03/12/2004, H.E.B., $26.50]

Ending Balance: 954.26
```

Lab 18D - 100 points

OBJECTIVE

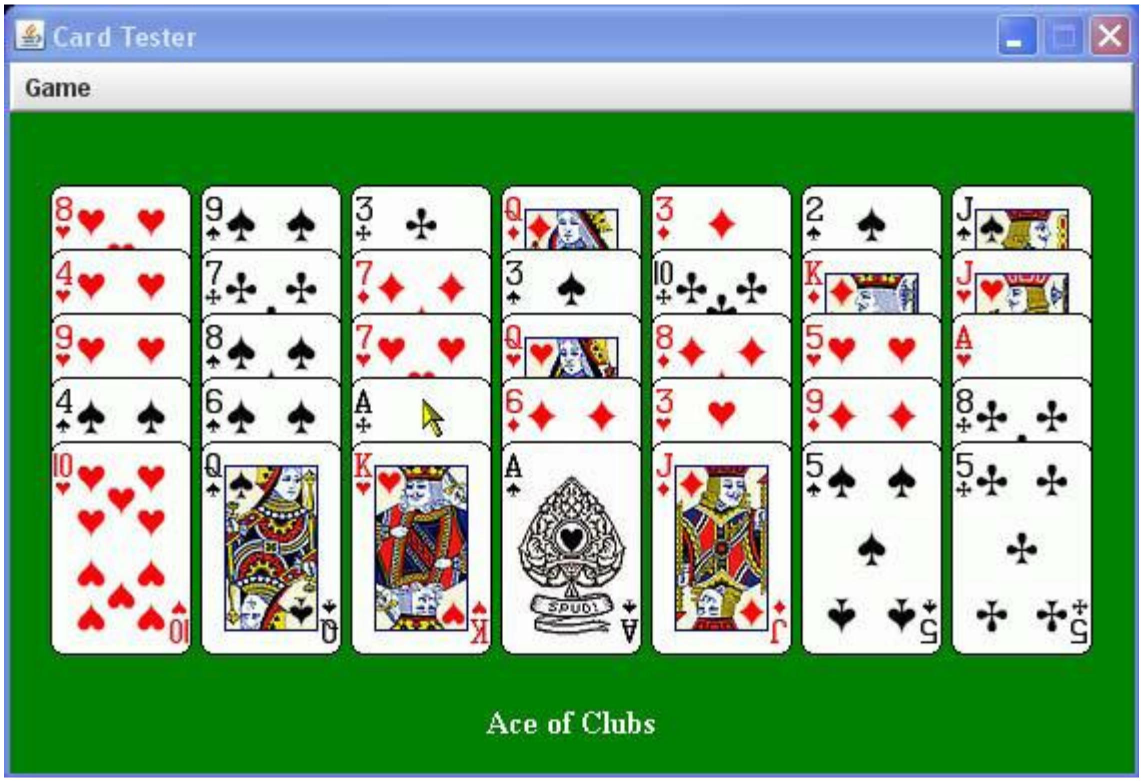
Playing cards are typically hand-sized piece of heavy paper or thin plastic. A complete set of cards is a pack or deck. A deck of cards is used for playing many card games. Every card in the deck has both a rank and a suit. Various formats of bitmap images are used in computer card games to represent the cards.

WAC called `Card`. Read ***Card.html*** for a detailed summary of the fields, constructors, and methods required for the `Card` class. When you have completed the `Card` class double click on ***CardTester.bat*** to test your class.

METHOD SUMMARY

- **constructor** – The constructor should receive three arguments: an instance of an `Image`[1], and two integers, `rank` and `suit`. Store all three arguments in the appropriate instance fields. The `Image` will be used when `drawCard` is called.
- Include `accessors` for rank and suit (i.e. `getRank` and `getSuit`).
- Cards are immutable. Once constructed they cannot be changed. Therefore no mutators are required.
- **drawCard** – receive three arguments: a `Graphics`[2] object, and two `ints` (`x` and `y` coordinate). Draw the cards image on the graphics context received as an argument at coordinates `x, y`.[3]
- **toString** – should return a String representation of the current card. (i.e. "Ace of Spades", "2 of Diamonds", "King of Clubs", etc.) [4]

[1] You will need to import **java.awt.Image**.
[2] You will need to import **java.awt.Graphics**.
[3] Images are drawn by calling the drawImage method of the Graphics class. For example, given a Graphics context **g** you can draw the Image **pic** by calling **g.drawImage(pic, x, y, null)** where **x** is the x coordinate and **y** is the y coordinate.
[4] Arrays can be very useful for displaying both the suit and the rank. For example, if you declare an array **String[] theSuit = { "Clubs", "Diamonds", "Hearts", "Spades" };** then indexing into the array using the suit variable would result in a String, i.e. theSuit[suit].



Ace of Clubs