Class 6 Notes

The keyword ***this***

Look at the one argument constructor for the Dice class:

```
public class Dice
{
        private int numDice;

        public Dice(int n)  // one arg constructor
         {
                numDice = n;
        }

        // other methods go here
}
```

It is possible to use the name *numDice* for the parameter. But, how would the assignment work?

```
public class Dice
{
        private int numDice;

        public Dice(int numDice)  // one arg constructor
         {
                numDice = numDice;
        }

        // other methods go here
}
```

Wow! numDice = numDice.
This will compile with no error but it will really do nothing.

It assigns the parameter numDice to itself.It does not assign the value in the parameter numDice to the class (global or instance) variable **numDice.**

You can reference the instance variable (**numDice) as**

```
    public class Dice
    {
        private int numDice;

        public Dice(int numDice)  // one arg constructor
         {
             this.numDice = numDice;
        }

        // other methods go here
    }
```

this is actually a reference whereby an object can refer to itself.
The reference *this* means “the current object.”

Now look at this small class:

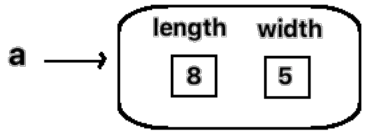
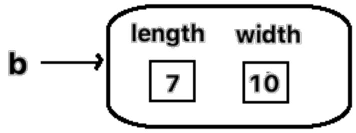
```
public class UseDice
{
   public static void main(String[] args)
   {
        Dice d = new Dice(2);  // calls the constructor
        ……..
    }
}
```

In the constructor this.numDice refers to the numDice of object d.

Example: a little tricky

```
public class Rectangle
{
        private int length, width;

        public  Rectangle()  // default constructor
        {
                Length = width = 1;
        }

        public Rectangle(int length, int width)  // one argument constructor
        {
                this.length = length;
                this.width = width;
        }

        public int area()
        {
                return length*width;
        }

        public Rectangle bigger(Rectangle r)
       {
                // returns a reference to the bigger of two Rectangles
                // bigger means bigger area
               // what is the code?
        }


        public static void main(String[] args)
        {
                Rectangle a = new Rectangle(8,5); // area is 40
                Rectangle  b = new Rectangle(7, 10); // area is 70

                Rectangle c = a.bigger(b);
                // c is a referencr to the bigger Rectangle – that is  Rectangle b
                System.out.println(“The larger area is “+ c.area());
        }
}
```

Here is what is going on:

Rectangle a = new Rectangle(8,5);	Rectangle b= new Rectangle(7,10);
	

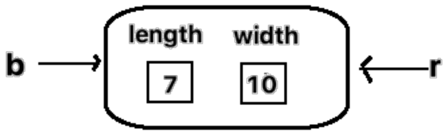
Look at the line:

```
Rectangle c = a.bigger(b);
```

Rectangle b is passed the method

```
public Rectangle bigger(Rectangle r)
```

b and r reference the same Rectangle object:



Here is how it works:

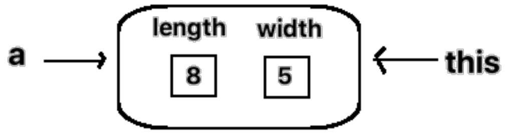
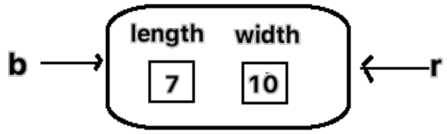
```
public Rectangle bigger(Rectangle r)
 {
        // returns a reference to the bigger of two Rectangles
        If (this.area() > r.area()
                return this;
        else return r;
}
```

The reference *this* refers to Rectangle a (the object that calls bigger)

You **can't** say

```
If (a.area() > r.area())
```

Because the method does not have a Rectangle a.
The within the method the picture would be

So this is a reference to the calling rectangle a.

Recap:

Object Oriented Programming:

- Encapsulation
- Inheritance
- Polymorphism

Encapsulation: The language feature where data and methods are bundled as a single entity called an object.

One more example:

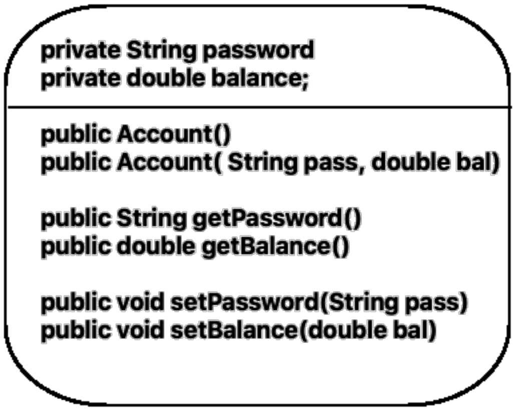
A bank maintains a list of accounts. For simplicity, we will assume an account consists of a password and a balance.

Write a program that simulates an ATM machine.
When a customer goes to the ATM he/she enters a password. If the password is recognized the customer has a choice:

Withdraw money
Deposit Money
Get the current balance
End the transaction

We will assume that a customer has at most five tries with the password.

We will make two classes: Account and Bank



```
private String password
private double balance;

public Account()
public Account( String pass, double bal)

public String getPassword()
public double getBalance()

public void setPassword(String pass)
public void setBalance(double bal)
```

Account

```
public class Account
{
  private String password;
  private double balance;

  public Account()                    // default constructor
  {
    password = "";
    balance = 0.0;
  }

  public Account(String pass , double bal)      // 2 argument construnctor
  {
    password = pass;
    balance = bal;
  }

  public String getPassword()
  {
    return password;
  }

  public double getBalance()
  {
    return balance;
  }

  public void setPassword(String p)
  {
    password = p;
  }

  public void setBalance(double b)
  {
    balance = b;
  }
}
```

The Account class is pretty simple.

The Bank class is a little more complicated

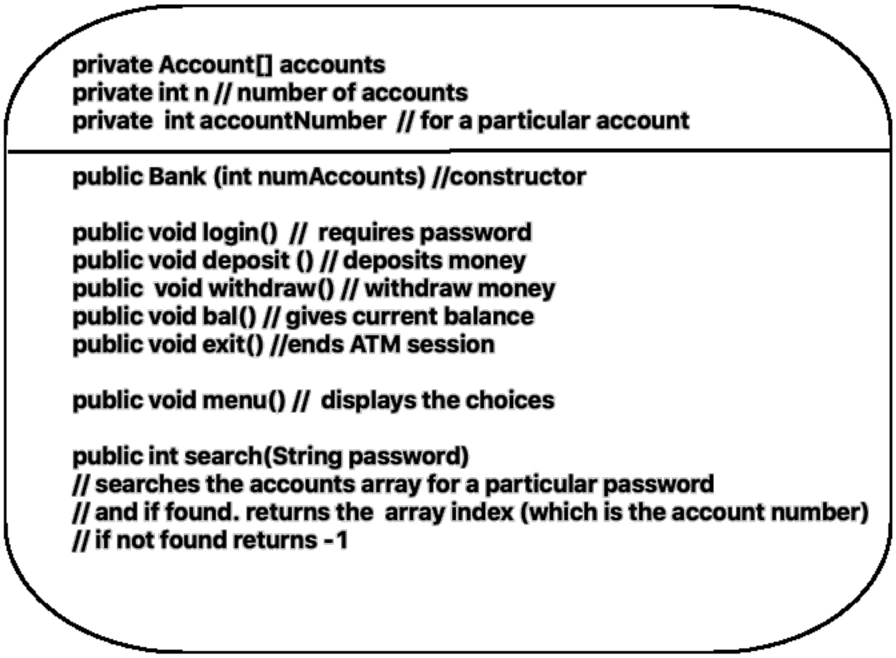
The Bank class will maintain a list of Account objects. That is an array.

It will fill the array from a file, say bank.txt

The file will consist of records in the form (password, balance)
For example:

11111 100 → password is 11111, balance is 100
22222 200
33333 300
44444 400

The class will have the following structure.



```
private Account[] accounts
private int n // number of accounts
private  int accountNumber  // for a particular account
```

```
public Bank (int numAccounts) //constructor

public void login()  //  requires password
public void deposit () // deposits money
public  void withdraw() // withdraw money
public void bal() // gives current balance
public void exit() //ends ATM session

public void menu() //  displays the choices

public int search(String password)
// searches the accounts array for a particular password
// and if found. returns the  array index (which is the account number)
// if not found returns -1
```

Bank

Notice that accounts is an array of objects.
The account number is the array index for a particular account.

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

public class Bank
{
 private Account[] accounts;  // array of all accounts
 private int n;                          // number of account
 int accountNumber = 0;       // for a particular user



 public Bank(int numAccounts) throws IOException   // constructor
 {
  // reads a file of bank accounts into the array accounts
  File f = new File("bank.txt");
  Scanner input = new Scanner(f);
  n =numAccounts;

  accounts = new Account[n];
  for (int i = 0; i < n; i++)
  {
     String pass = input.next();                    // read the password
     double bal = input.nextDouble();          // read the current balance
     accounts[i] = new Account(pass, bal);  // make Account object and add to array
  }
  input.close();
 }





 public int search(String pass)      // search array for the password
 {
  for (int i = 0; i < n; i++)
    if (pass.equals(accounts[i].getPassword())) // password found
          return i;

   return -1; // password was not found
  }
```

```
public void login() throws IOException
{
 int count = 0;                              // number of password tries
 Scanner input = new Scanner(System.in);

 do
 {
    count++; // increment number of password tries
    System.out.print("Enter password: ");
    String password = input.next();
    accountNumber = search(password);
 }while (count <= 5 &&  accountNumber == -1);

 if (count > 5  || accountNumber == -1) //too many tries
 {
      System.out.println("Invalid password");
      System.exit(0);
 }
 else  // valid password
 {
      menu();
 }
}
```

```
public void menu()  throws IOException
{

 System.out.println("Enter choice");
 System.out.println(" 1 for deposit");
 System.out.println(" 2 for withdrawal");
 System.out.println(" 3 for balance");
 System.out.println(" 4 for exit");

 System.out.println();
 System.out.println("------------------");
 System.out.println();

 Scanner input = new Scanner(System.in);
 int choice = input.nextInt();

// should really check for invalid input here

 switch(choice)
 {
  case(1) : deposit(); break;
  case(2) : withdrawal(); break;
  case(3): bal(); break;
  case(4): exit(); break;
 }
}



 public void deposit() throws IOException
 {
  Scanner input = new Scanner(System.in);
  System.out.print("Enter deposit: ");
  double deposit = input.nextDouble();

  double newBalance = accounts[accountNumber].getBalance()+ deposit;

  accounts[accountNumber].setBalance(newBalance);

  menu();
 }
```

```
public void withdrawal() throws IOException
{
    Scanner input = new Scanner(System.in);
    System.out.print("Enter withdrawal amount");
    double amount = input.nextDouble();

    double currentBalance = accounts[accountNumber].getBalance();

    if (amount > currentBalance)
        System.out.println("Invalid amount.  Balcance is "+ currentBalance);
    else
          accounts[accountNumber].setBalance(currentBalance - amount);

   menu();
}
```

```
public void bal() throws IOException
{
  System.out.println("Balance: "+ accounts[accountNumber].getBalance());
  menu();
 }
```

```
public void exit() throws IOException
{
   // updates the file with the current information
   System.out.println("Session over");
   File f = new File("bank.txt");  // this is a little dangerous, why
   PrintWriter pw = new PrintWriter(f);
   for (int i =0; i <n; i++)
     pw.println(accounts[i].getPassword()+ "  "+ accounts[i].getBalance());
   pw.close();

}
```

```
  public static void main(String[] args)  throws IOException
  {
    Bank b = new Bank(5);
    b.login();
  }

}
```

Notice throws IOException is on just about every method

main() calls login() and login uses files → both need throws IOException

menu calls exit() and exit() uses files → both need throws IOException

deposit () calls menu() which calls exit() → deposit() also needs throws IOException

etc.

Next topic : Inheritance