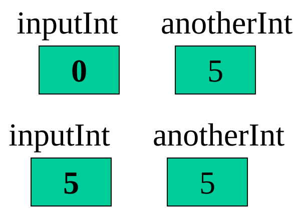
Week 5

Primitives and References
Classes and Objects
Creating your own classes
Methods
Instance variables
Data encapsulation
Constructors
StringTokenizer class
Assignment 2

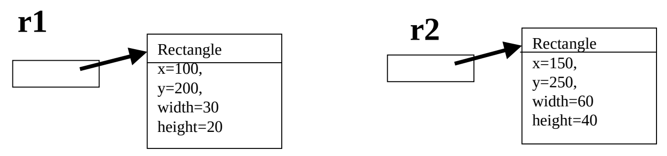
Assignment Primitives

```
int inputInt = 0;
int anotherInt = 5;
...
inputInt = anotherInt;
```

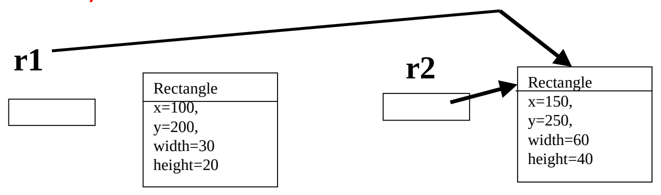


Assignment References

```
Rectangle r1 = new Rectangle(100,200,30,20);
Rectangle r2 = new Rectangle(150,250,60,40);
```



```
r1 = r2;
```



3

Sample program

```
import java.awt.*;
public class RectangleObjects {
  public static void main (String[] args) {
    Rectangle r1 = new Rectangle(100,200,30,20);
    Rectangle r2 = new Rectangle(150,250,60,40);
    r1 = r2;
    r1.translate(100,50);
     System.out.println("r1: x= "+r1.getX()+" y= "+r1.getY()+
                    " w="+r1.getWidth()+" w="+r1.getHeight() );
     System.out.println("r2: x= "+r2.getX()+" y= "+r2.getY()+
                    " w="+r2.getWidth()+" w="+r2.getHeight() );
  }
}
```

output

```
r1:  x=250.0  y = 300.0  w =  60.0  h = 40.0
r2:  x=250.0  y = 300.0  w =  60.0  h = 40.0
```

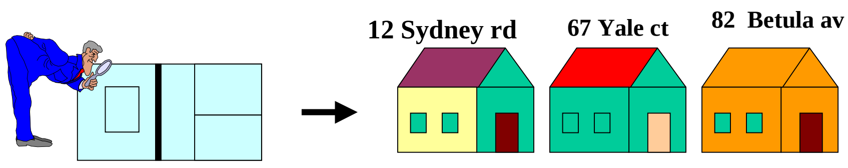
What will be the output if we replace r1 = r2; with:

(a) r2 = r1;

(b) r1.setBounds(r2);

4

Classes and Objects



We say that an object is an instance of a class.

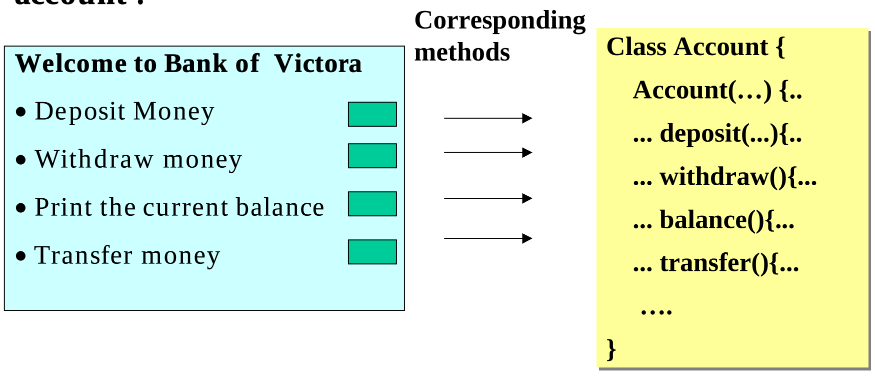
A class is a template, a model from which instances can be created

Recall we created the ConsoleReader class to facilitate input.

5

Deciding Object Behavior for our own classes

- Suppose we want to create a class to model a bank account.
- What kind of operations can you carry out with a bank account ?

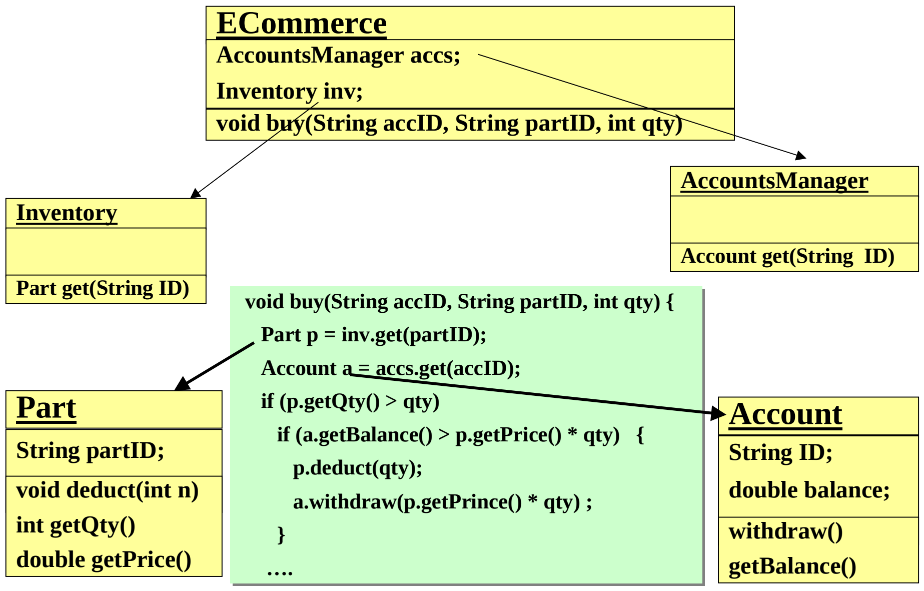


Welcome to Bank of Victora

- Deposit Money 🟩
- Withdraw money 🟩
- Print the current balance 🟩
- Transfer money 🟩

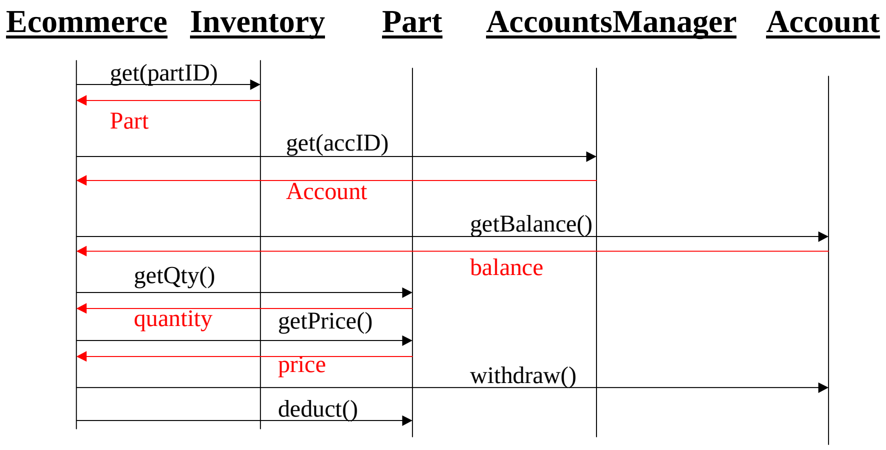
```
Class Account {
   Account(…) {..
   ... deposit(...){..
   ... withdraw(){...
   ... balance(){...
   ... transfer(){...
    ….
}
```

Objects Communicate by sending messages



```
void buy(String accID, String partID, int qty) {
  Part p = inv.get(partID);
  Account a = accs.get(accID);
  if (p.getQty() > qty)
    if (a.getBalance() > p.getPrice() * qty)   {
      p.deduct(qty);
      a.withdraw(p.getPrince() * qty) ;
    }
  ....
```

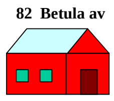
Object Interaction
buy() method of ECommerce



8

Constructing an Account object

Bill requested the builder to construct a Rochester (type) at 82 Betula avenue with red brick and blue tiles.

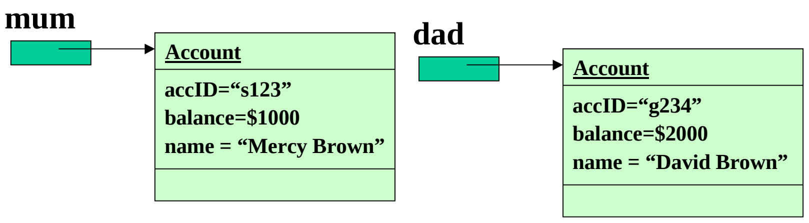


We can use Account constructor to create an Account objects specifying the arguments as in

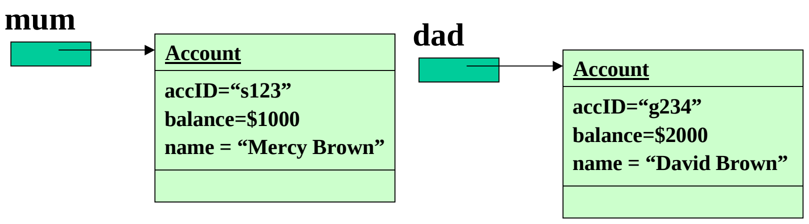
```
Account mum = new Account("s123","Mercy Brown",1000);
Account dad = new Account("g234","David Brown",2000);
```

Account constructor

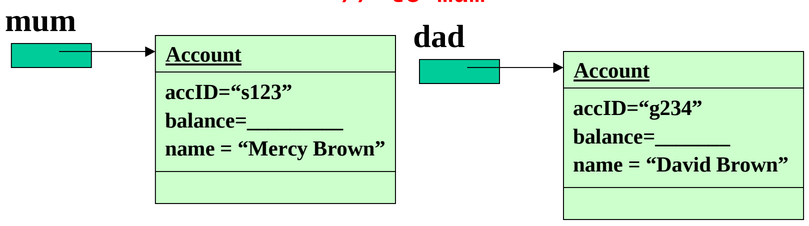
Arguments(relevant information)



What can we do with them ? Operations.



```
mum.withdraw(100);       // mum's withdrawing $100
dad.deposit(150);        // dad's depositing $150
dad.transfer(mum,500);   // dad's transferring $500
                         // to mum
```

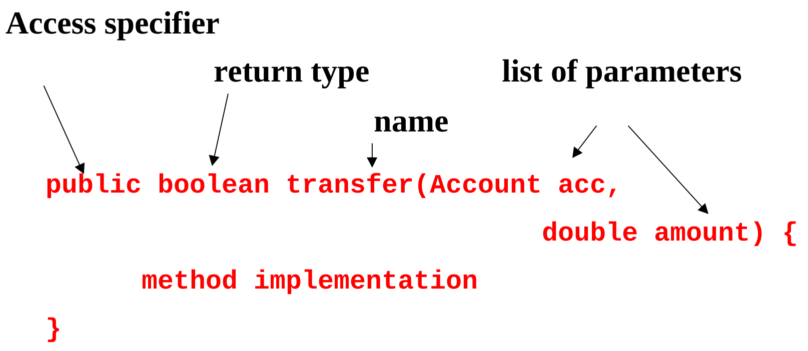


These operations need methods

```
public void deposit(double amount) {
       method implementation
}
public boolean withdraw(double amount) {
       method implementation
}
public boolean transfer(Account acc, double amount) {
       method implementation
}
public double getBalance() {
       method implementation
}
public double getID() {
       method implementation
}
```

11

Parts of a methods



```
public boolean transfer(Account acc,
                              double amount) {
       method implementation
}
```

12

What makes one Account object different from another ?

Each instance stores the values reflecting its current state (such as balance, ID , name)

From the operations we have identified we should have at least three instance variables.

```
public class BankAccount
{  …
   private String name;
   private double balance;
   private String accID;
}
```

private

(not accessible by class users)

13

Accessing balance ...

```
Account mum = new Account("s123",
                    "Mercy Brown",1000);
// Error balance is private
double d = mum.balance;
```

But you can call the public getBalance() method to inquire about the balance:

```
// OK
double d = mum.getBalance();
```

14

Why private ?

Suppose we created a class to model a switch which has two possible states on and off as in:

```
class switch {
    switch() {state = 0; }
    public setOff() { state = 0;}
    public setOn() {state = 1; }
    public int state;
}
```

```
int on = 1;
switch a = new
switch();
a.state = on;
...
```

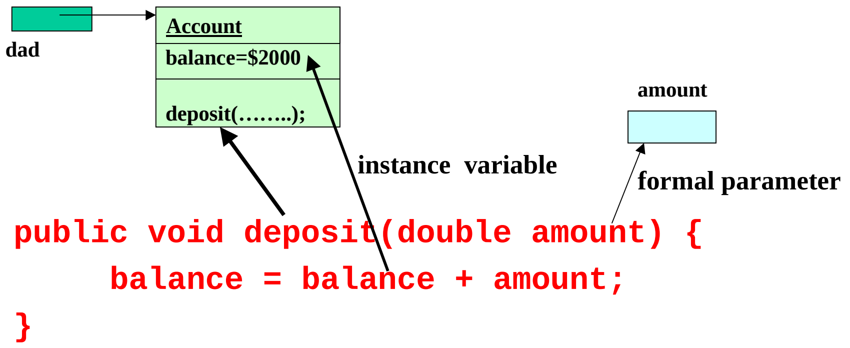
- Can we now change state to a boolean variable (efficient) ?
- We cannot because class users have assumed it is an int and written code.
- So data should always be hidden (private) - Encapsulated.

15

Implementing Methods

To deposit $150 to dad we used:

```
dad.deposit(500); //deposit() of dad called
```



```
public void deposit(double amount) {
      balance = balance + amount;
}
```

Formal parameters are used for passing arguments.
They come into existence only when method is invoked.

16

Implementing Methods

To withdraw $100 from mum we used:

```
mum.withdraw(100); // withdraw() of mum called
```

withdraw() reduces the balance and to return true if funds are sufficient and false otherwise.

```
public boolean withdraw(double amount){
   boolean result = false;

   if ( balance > amount ) {
            result = true;
            balance -= amount;
   }
   return result;
}
```

local variable cannot be used outside the method

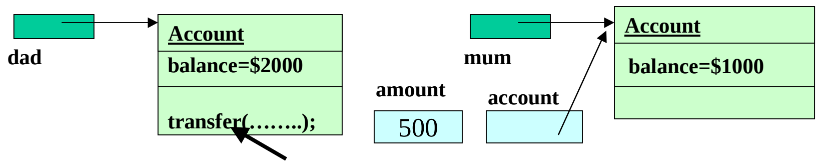
formal parameter is used for passing arguments

Local variables are used within a method. Like formal parameters they come into existence only when that method is invoked.

Implementing Methods

To deduct money from dad's balance and to credited to mum's use the statement below. It fisrt verifies dad has sufficient funds.

```
dad.transfer(mum,500); //transfer() of dad called
```



```
public boolean transfer(Account account,
                        double amount) {
   if (balance >= amount) {
      balance = balance - amount;
      account.balance += amount;
      return true;
   }
   else return false;
}
```

18

Accessor & Mutator methods

A method that accesses an object and returns some information about it, without changing the object is called an *accessor* method.

```
public double getBalance() {
      return balance;
}
public String getID() {
      return accID;
}
public String getName() {
    return name;
}
```

Methods that modify the state of an object are called *mutator* methods. Example deposit(), withdraw().

19

Constructors

- **Initializes the instance variable of an object.**
- **Has the same name as the class generally public.**
- **But unlike other methods have no return types.**
- **If no constructor provided Java will provide one with all instance variables set to default values.**
- **Account constructor is invoked twice below**

Account mum = new Account("s123","Mercy Brown",1000);

Account dad = new Account("g234","David Brown",2000);

```
public Account(String accountID,
      String accountName, double amount){
            accID = accountID;
            name = accountName;
            balance = amount;
}
```

20

Can I have more than 1 constructor ?

- **Yes provided the method signature is different.**
- **The one below only takes two arguments. As no value is passed for balance it is set to 0.**
- **When called with 2 string arguments as shown below this constructor will be invoked.**

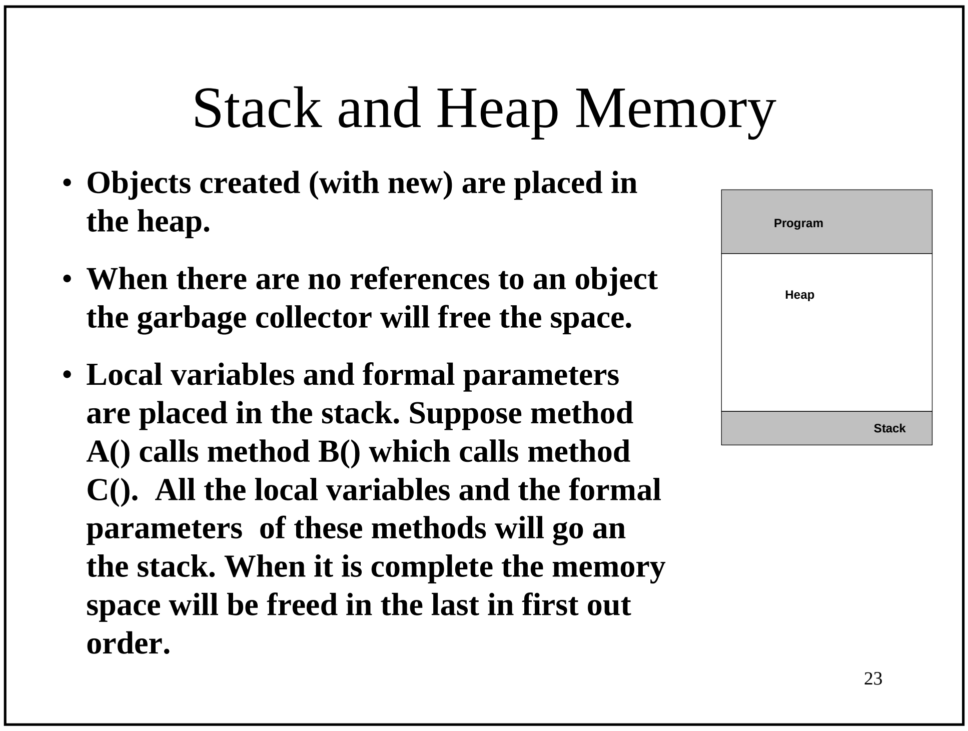
```
Account poor = new Account("s123", "Mercy Brown");

public Account(String accountID,
                        String accountName){
      accID = accountID;
      name = accountName;
      balance = 0.0;
}
```

Testing the Account class

The TestAccount class in a separate file (TestAccount.java) creates two Account objects and performs the operations discussed above.

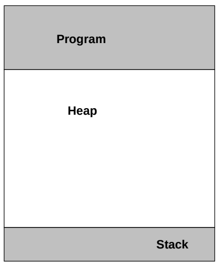
The final balance for mum and dad should be $1400 and dad $1,650 respectively.

```
public class TestAccount  {
  public static void main(String args[]) {
      Account mum = new Account("s123","Mercy Brown",1000.0);
      Account dad = new Account("g234","David Brown",2000.0);
      mum.withdraw(100);
      dad.deposit(150);
      dad.transfer(mum,500);
      System.out.println("mum bal = "+mum.getBalance());
      System.out.println("dad bal = "+dad.getBalance());
  }
}
```

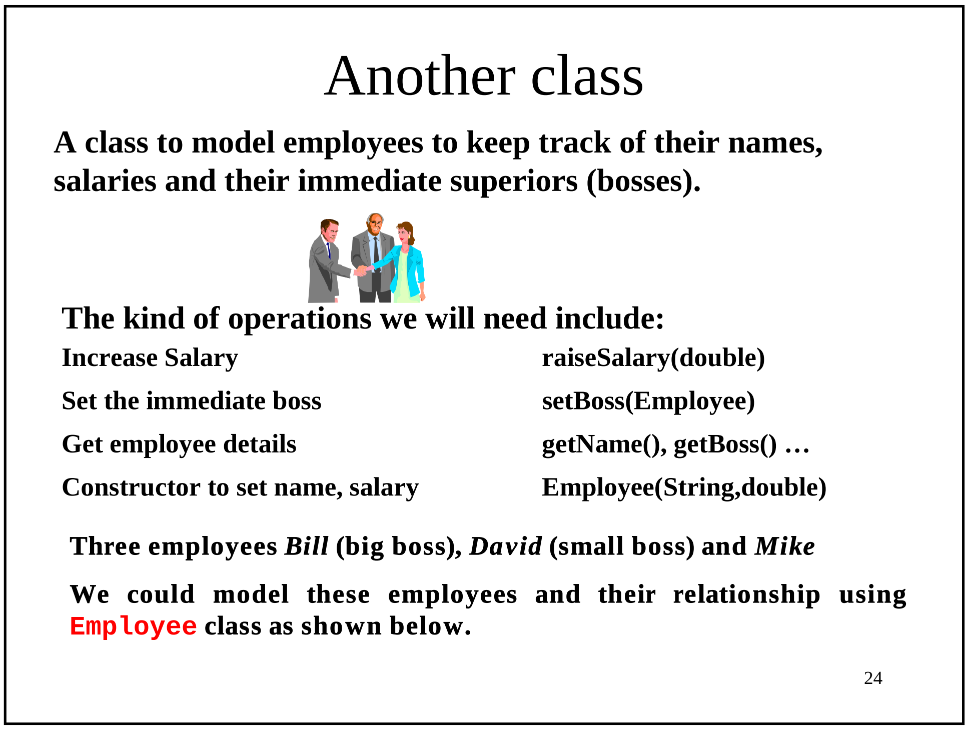


Stack and Heap Memory

- Objects created (with new) are placed in the heap.
- When there are no references to an object the garbage collector will free the space.
- Local variables and formal parameters are placed in the stack. Suppose method A() calls method B() which calls method C(). All the local variables and the formal parameters of these methods will go an the stack. When it is complete the memory space will be freed in the last in first out order.



23



Another class

A class to model employees to keep track of their names, salaries and their immediate superiors (bosses).

The kind of operations we will need include:

Increase Salary	raiseSalary(double)
Set the immediate boss	setBoss(Employee)
Get employee details	getName(), getBoss() …
Constructor to set name, salary	Employee(String,double)

Three employees *Bill* (big boss), *David* (small boss) and *Mike*

We could model these employees and their relationship using `Employee` class as shown below.

24

```
public class TestEmployee   {
  public static void main(String args[]) {
       Employee bill = new Employee("Bill Gates",100000.0);
       Employee david = new Employee("David Paul",80000.0);
       Employee mike = new Employee("Mike Mogan",50000.0);
       david.setBoss(bill);
       mike.setBoss(david);
       mike.raiseSalary(10000);
       david.raiseSalary(20000);
       // printing details of mike
       System.out.print("name = " + mike.getName());
       System.out.print(" Salary = " + mike.getSalary());
       Employee boss = mike.getBoss();
       // if mike has a boss print his details
        if (boss != null)
              System.out.println("Reports to "+boss.getName());
        else System.out.println();
  }
}
```

```
public class Employee  {
   public Employee(String empName, double empSalary)   {
name = empName;
              salary = empSalary;
  }
  private String name;
  private double salary;
  private Employee boss;

  public void raiseSalary(double amount) {
        salary = salary + amount;
  }

  public void setBoss(Employee empBoss) {
        boss = empBoss;
  }
  public double getSalary() {
        return salary;
  }

  public String getName() {
        return name;
  }

  public Employee getBoss()   {
       return boss;
  }
}
```

Using the StringTokenizer class

What if we have to read more than one integer from a line

```
Please enter all assignment marks for all students
8 5 9
7 8 9 5
6 7 4
```

- **To extract one token at a time from a string we can use the StringTokenizer class.**
- **Method `countTokens()` to find the number of tokens.**
- **Method `nextToken()` to get the next token**

```
//  reads and prints int in input
//  until a null string is entered

import java.io.*;
import java.util.StringTokenizer;

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

            int count = 0;

            // create a text input stream
            BufferedReader stdin = new BufferedReader
                         (new InputStreamReader (System.in));
            // to do the splitting we need a new instance
            // of StringTokenizer
            StringTokenizer inReader;

            System.out.println("Enter int numbers " +  "separated by spaces");
            System.out.println("Hit return by itself " +  "to finish");

            // read the next line and tokenize it
            inReader = new StringTokenizer (stdin.readLine());
```

29

```
              while (inReader.countTokens() > 0)  { // still input
                           count = inReader.countTokens();
                           while (count > 0)  {  // process this line
                                System.out.println("The next numbers is: " +
                                Integer.parseInt(inReader.nextToken()));
                                count--;
                           }  // finish processing line

                           //  read the next line and tokenize it
                           inReader = new StringTokenizer(stdin.readLine());
              }
        }
  }
```

30

Copy and modify the Account class calling it SAccount to model a savings account. Assume savings account holders are required to maintain a minimum amount (which may vary from customer to customer). Hence, the constructor should take an additional argument, the minimum amount to be maintained. An additional data member should be introduced in the class to hold this value. The withdraw() method should be modified to reflect the minimum amount, ie. the account balance should always be maintained above minimum amount. This class should also provide an additional method addInterest(), for adding monthly interest given the interest rate. You are also required to provide an additional method (accessor) getMinAmount(), which will return the minimum amount that account holder is required to maintain.

Test your SAccount class with the driver below. Notice that the withdraw() and transfer() methods should fail (return false) if withdrawing the specified amount will leave the balance less than the minimum amount to be maintained. Verify that final balances printed are the correct values.

31

```
public class TestSAccount{
  public static void main(String args[]) {
    // Note both accounts must maintain minimum amount $100
    SAccount mum = new SAccount("s123","Mercy Brown",200.0,100.0);
    SAccount dad = new SAccount("g234","David Brown",2000.0,100.0);

    //  withdraw() should return false this time
    if ( mum.withdraw(150) == false)
          System.out.println("mum unable to withdraw - first time");
    if (dad.transfer(mum,100) == false)
          System.out.println("dad unable to transfer");
    //  withdraw() should return true this time
    if ( mum.withdraw(150) == false)
          System.out.println("mum unable to withdraw - second time");
```

32

```
    mum.addInterest(0.5);            // (Jan. interest rate = 0.5%)
    dad.addInterest(0.5);            // (Jan. interest rate = 0.5%)
    dad.deposit(150);
    mum.addInterest(0.55);           // (Feb. interest rate = 0.55%)
    dad.addInterest(0.55);           // (Feb. interest rate = 0.55%)

    System.out.println("mum final bal = " + mum.getBalance());
    System.out.println("dad final bal = " + dad.getBalance());
  }
}
// (*** You are required to submit this ***)
```