Another OO Example

EXTRA MATERIAL

** steps to write OO programs
** overloading ** overriding

Chapter 2 (sections 2.1–2.3; 2.5–2.8; 2.10–2.11) – "Core Java" book
Chapters 2-4 – "Head First Java" book
Chapter 5+10+11 (sections 5.8, 10.11, 11.4-11.5) – "Introduction to Java Programming" book
Chapter 3 – "Java in a Nutshell" book

These slides are left as practice and self-study.

Extra example: the bank account

- Design a class for a bank account.

BankAccount
account number account name balance
deposit withdraw

UML to Java code

attributes

account number
account name
balance

instance variables

```
int accNo
String accName
double balance
```

operations

deposit
withdraw

methods

```
deposit(double amount)
withdraw(double amount)
```

In Java, instance variables are used to define an object's states (or attributes) and methods are used to define its behaviour.

A Java class: general template

```
class ClassName {
    // instance variables
    // constructors
    // accessors (or getters)
    // mutators (or setters)
    // service methods
}
```

Step 1. Instance Variables

```
public class BankAccount {
  private int accNo;
  private String accName;
  private double balance;

  // other code to add ...


}
```

Using **`private`** for information hiding.

Step 2. Constructors

```
public class BankAccount{
   private int accNo;
   private String accName;
   private double balance;

   public BankAccount(int accNo, String accName){
      this.accNo = accNo;
      this.accName = accName;
      this.balance = 0.0;
   }

   public BankAccount(String accName, int accNo){
      this.accNo = accNo;
      this.accName = accName;
      this.balance = 0.0;
   }

   // other code to add ...
}
```

Constructor has the same name as the class. User-defined constructor assigns values from arguments.

They are different constructors.

Step 3. Accessors (getters) and Step 4. Mutators (setters)

```
public class BankAccount {
   // instance variables
   // constructors

   public int getAccountNo() { return accNo; }

   public String getAccountName() { return accName; }

   public double getBalance() { return balance; }

   // other code to add ...
}
```

Provide them only if you allow others to retrieve the states.

```
   public class BankAccount {
      // instance variables
      // constructors
      // accessors (getters)

      public void setAccountName(String accName) {
        this.accName = accName;
      }

      // other code to add ...
   }
```

The account number cannot be set. Directly setting the balance is not allowed; balance changes through `deposit()` and `withdraw()`.

Step 5. Service methods

- Service methods are used to interact with the data in the object and to change the state of the object.
- In the **BankAccount** example, we can change the state of the balance by making a deposit or a withdraw.
  - In this case, the amount will be passed.

```
public class BankAccount {

    // instance variables
    // constructors
    // accessors (getters)
    // mutators (setters)

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

    public void withdraw(double amount) {
      balance = balance - amount;
    }

    // other code to add ...
}
```

Step 6. `toString()` method (1/3)

- To print primitive data (e.g. **`int`**, **`double`**, **`char`**) and **`String`** we use:

```
System.out.println(variableName);
```

- Is it possible to print out an object?
  - What happens if we try to print an object like this?

```
BankAccount myAccount = new BankAccount(11111111,
                                        "John");
System.out.println(myAccount);
```

- There is an instance of the class **`BankAccount`**, along with an object reference.
- The compiler knows where the object is and what is stored in it, but cannot print it out correctly.

BankAccount@119c082

This looks a bit like an email address! What is it?

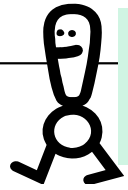
- Because we have not told the compiler how to represent it!

Step 6. `toString()` method (2/3)

- Methods like **`println()`** or **`print()`** want a **`String`** representation of the object to print.
- To represent the object as a **`String`**, we need to implement the **`toString()`** method.
  - We will actually override the **`toString()`** method defined in the **`Object`** class.
  - This method returns a **`String`** representation of the object.

`toString()` method for our **`BankAccount`** class

```
public String toString() {
    return "Account number: " + accNo + "\n"
           + "Account name: " + accName + "\n"
           + "Balance: " + balance ;
}
```



This method must be named **`toString()`** and it must return a **`String`** type.

Step 6. `toString()` method (3/3)

```
public class BankAccount {
    // instance variables
    // constructors
    // accessors (getters)
    // mutators (setters)
    // deposit method
    // withdraw method
    public String toString() {
     return "Account number: " + accNo + "\n"
            + "Account name: " + accName + "\n"
            + "Balance: " + balance ;
    }
}
```

With the `toString()` method, the object can be printed out with a user-defined format.

```
BankAccount myAccount = new BankAccount(11111111, "John");
System.out.println(myAccount);
```

Account number: 11111111
Account name: John
Balance: 0.0

BankAccount class (in full)

```
public class BankAccount{
   private int accNo;
   private String accName;
   private double balance;

   public BankAccount(int accNo, String accName) {
     this.accNo = accNo;
     this.accName = accName;
     this.balance = 0.0;
   }

   public BankAccount(String accName, int accNo) {
     this.accNo = accNo;
     this.accName = accName;
     this.balance = 0.0;
   }

   public int getAccNo() {
     return accNo;
   }

   public String getAccName() {
     return accName;
   }

   public double getBalance() {
     return balance;
   }

   public void setAccName(String accName) {
     this.accName = accName;
   }

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

   public void withdraw(double amount) {
     balance = balance - amount;
   }

   public String toString() {
     return "Account number: " + accNo
       + "\n" + "Account name: " + accName
       + "\n" + "Balance: " + balance;
   }
}
```

Step 7. A test class

```
public class BankAccountTest {
   public static void main(String[] args) {
      BankAccount acc1 = new BankAccount(23142635, "John Smith");
      System.out.println(acc1);
      acc1.deposit(500);
      acc1.withdraw(100);
      System.out.println(acc1);

      BankAccount acc2 = new BankAccount("Tom Will", 38472638);
      System.out.println(acc2);
      acc2.deposit(3000);
      acc2.withdraw(400);
      System.out.println(acc2);
   }
}
```

```
Account number: 23142635
Account name: John Smith
Balance: 0.0
Account number: 23142635
Account name: John Smith
Balance: 400.0
Account number: 38472638
Account name: Tom Will
Balance: 0.0
Account number: 38472638
Account name: Tom Will
Balance: 2600.0
```

Method Overloading

- Java allows several methods to be defined with the same name, as long as they have different sets of parameters.
- The compiler resolves which particular method is required by examining the signature of the method – its name and the types and sequence of its parameters.
- The return type is NOT used to differentiate methods, so you cannot declare two methods with the same signature even if they have a different return type.
- Examples:

```
public void deposit(double amount, boolean cheque) {
  if (cheque == false) { balance = balance + amount; }
  else {
    // code to be added
  }
}
```

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

Improving the code …

1. The variable **accNo**: **int** or **String**?
   - Consider the account number **00112612**
2. A better **withdraw** method: do not allow overdraft
3. An even better **withdraw()** method: set up overdraft limit
4. Print some user friendly messages

In our **BankAccount** example …

- A better **withdraw()** method; it does not allow a withdrawal if **amount > balance**
- An even better **withdraw()** method
  - How about setting up an overdraft limit? Try at home …

```
public void withdraw(double amount) {
   if (balance >= amount) {
      balance = balance - amount;
   }
}
```