Inheritance, overloading and overriding

- **Recall**
  - with inheritance the behavior and data associated with the child classes are always an extension of the behavior and data associated with the parent class

- **In a child class you can**
  - redefine a method's implementation (override)
    - a method that is inherited by the parent, and the child class wants to change its behavior
  - define new methods with the same method *name* but different arguments (overload)
    - different tasks are performed by each method but they share the same method name
The Bank Account example

• Accounts must have
  – current balance
  – name of account holder
  – a withdraw method
  – a deposit method

• Current accounts
  – have a maximum withdraw amount
    • you cannot withdraw more than $200 in one transaction

• Savings accounts
  – have a minimum balance that they need to maintain at all times.
The Bank Account example

- Shared behavior and data between Savings and Checking
  - Data
    - current balance
    - name of account holder
  - Behavior (method names and implementation)
    - accessors for common data
    - deposit
  - Behavior (method names without behavior)
    - withdraw
    - display
The Bank Account example

- Account is a generalized idea
- What actually exists in the banking model are savings and checking accounts.
  - both are accounts with specialized operations on them. you can refer to them as accounts but you are using them according what a savings and/or a checking account can do.
- Generalized ideas (i.e. Account) can be directly mapped in java as
  - abstract classes
  - interfaces.
The Bank Account with abstract classes

**Account**

- double balance
- String name

+getBalance():double
+getName():String
+setName(String):void
+setBalance(double):void
+deposit(double):boolean
+withdraw(double):boolean
+display()

**abstract class**

**abstract methods**

- Abstract classes *cannot* be instantiated
  - there is no constructor!
- Abstract methods *must* be implemented by subclasses of Account
  - there is no body for withdraw and display inside Account
The Bank Account with abstract classes

Account

double balance
String name

+getBalance():double
+getName():String
+setName(String):void
+setBalance(double):void
+deposit(double):boolean
+withdraw(double):boolean
+display()

Checking

Savings

double minimumBalance

+getMinBal():double
+setMinBal(double):void
The Bank Account with abstract classes

<table>
<thead>
<tr>
<th>Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>double balance</td>
</tr>
<tr>
<td>String name</td>
</tr>
</tbody>
</table>

+getBalance():double
+getName():String
+setName(String):void
+setBalance(double):void
+deposit(double):boolean
+withdraw(double):boolean
+display()

```java
abstract public class Account {
    double balance;
    String name;

    public double getBalance() {
        return balance;
    }

    public void setBalance(double val) {
        balance = val;
    }

    public String getName() {
        return name;
    }

    public void setName(String aName) {
        name = aName;
    }

    public boolean deposit(double amount) {
        balance = balance + amount;
        return true;
    }

    abstract public boolean withdraw(double amount);
    abstract public void display();
}
```
The Bank Account with abstract classes

```
public class Checking extends Account{
    Checking(String name, double amount){
        this.name = name;
        if (amount > 0){
            this.balance = amount;
        } else {
            // error reporting code omitted
            this.balance = 0;
        }
    }
    public boolean withdraw(double amount){
        if (amount > 0 && amount <= balance) {
            balance = balance - amount;
            return true;
        } else if (amount > balance){
            // error reporting code omitted
            return false;
        } else {
            // error reporting code omitted
            return false;
        }
    }
    public void display(){
        System.out.println("  * * * * * *  Current Account Details 
        " + this.getName());
        System.out.println("  Name: " + this.getName());
        System.out.println("  Current Balance: "+
        this.getBalance());
        System.out.println("\t\t\t\t  * * * * * *  Current Account 
        Details ****** ");
    }
}
```
The Bank Account with abstract classes

<table>
<thead>
<tr>
<th>Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>double balance</td>
</tr>
<tr>
<td>String name</td>
</tr>
<tr>
<td>+getBalance():double</td>
</tr>
<tr>
<td>+getName():String</td>
</tr>
<tr>
<td>+setName(String):void</td>
</tr>
<tr>
<td>+setBalance(double):void</td>
</tr>
<tr>
<td>+deposit(double):boolean</td>
</tr>
<tr>
<td>+withdraw(double):boolean</td>
</tr>
<tr>
<td>+display()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>double minimumBalance</td>
</tr>
<tr>
<td>+getMinBal():double</td>
</tr>
<tr>
<td>+setMinBal(double):void</td>
</tr>
</tbody>
</table>

```java
public class Savings extends Account{
    double minimumBalance;
    Savings(String name, double amount, double minBalance){
        this.name = name;
        if (amount > 0){
            this.balance = amount;
        } else {
            // error reporting code omitted
            this.balance = 0;
        }
        if (minBalance > 0){
            this.minimumBalance = minBalance;
        } else {
            // error reporting code omitted
            this.minimumBalance = 0;
        }
    }
    public void setMinBal(double newBal){
        minimumBalance = newBal;
    }
    public double getMinBal(){
        return minimumBalance;
    }
    public boolean withdraw(double amount){
        // code omitted
    }
    public void display(){
        // code omitted
    }
```
Overloading the constructor

```java
public class Savings extends Account{
    double minimumBalance;

    Savings(String name, double amount, double minBalance){
        this.name = name;
        if (amount > 0){
            this.balance = amount;
        } else {
            // error reporting code omitted
            this.balance = 0;
        }
        if (minBalance > 0){
            this.minimumBalance = minBalance;
        } else {
            // error reporting code omitted
            this.minimumBalance = 0;
        }
    }

    Savings(String name, double amount){
        this.name = name;
        if (amount > 0){
            this.balance = amount;
        } else {
            this.balance = 0;
        }
        this.minimumBalance = 0;
    }
}
```

Distinguish the constructor by number of arguments and types for each argument.
Overloading the constructor

```java
public class Savings extends Account{
    double minimumBalance;

    Savings(String name, double amount,
             double minBalance){
        this.name = name;
        if (amount > 0){
            this.balance = amount;
        } else {
            // error reporting code omitted
            this.balance = 0;
        }
        if (minBalance > 0){
            this.minimumBalance = minBalance;
        } else {
            // error reporting code omitted
            this.minimumBalance = 0;
        }
    }
}

public class Main{

    public static void main(String[] args){
        Savings mySavings =
            new Savings("John",100.00,50.00);
        Savings anotherSavings =
            new Savings("Mary",200.00);
    }
}
```
Overloading a method

```java
public class Savings extends Account{
    double minimumBalance;

    public void display(){
        System.out.println("\n *****
Savings Account Details ***** ");
        System.out.println(" Name: "+
        this.getName());
        System.out.println(" Current Balance: "+
        this.getBalance());
        System.out.println(" Minimum Balance: "+
        this.getMinBal());
        System.out.println("\t\t\t\t *****
Savings Account Details *****\n ");
    }

    public void display(String date){
        System.out.println(date+"your balance is "+this.getBalance());
    }
}
```
Bank account with interfaces only!

- Interfaces in java define sets of operations that the type **must** implement.

<table>
<thead>
<tr>
<th>Checking</th>
<th>Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>double balance</td>
<td>double minimumBalance</td>
</tr>
<tr>
<td>String name</td>
<td>double balance</td>
</tr>
</tbody>
</table>

```
<<IAccount>>
+getBalance():double
+getName():String
+setName(String):void
+setBalance(double):void
+deposit(double):boolean
+withdraw(double):boolean
+display()
```

double minimumBalance
double balance
String name

+getMinBal():double
+setMinBal(double):void
Bank account with interfaces only!

- There is no implementation inside interfaces!
Bank account with interfaces only!

<table>
<thead>
<tr>
<th>Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>double minimumBalance</td>
</tr>
<tr>
<td>double balance</td>
</tr>
<tr>
<td>String name</td>
</tr>
</tbody>
</table>

+getMinBal():double
+setMinBal(double):void

```java
public class Savings implements IAccount {
    double minimumBalance;
    double balance;
    String name;

    // Same as Slide 7
    public double getBalance() { }
    public void setBalance(double val) { }
    public String getName() { }
    public void setName(String aName) { }
    public boolean deposit(double amount) { }

    // Same as Slide 9
    Savings(String name, double amount) { }
    Savings(String name, double amount, double minBalance) { }
    public void setMinBal(double newBal) { }
    public double getMinBal() { }
    public boolean withdraw(double amount) { }
    public void display() { }
}
```
**Bank account with interfaces only!**

```java
public class Checking implements IAccount {
    double minimumBalance;
    double balance;
    String name;

    // Same as Slide 7
    public double getBalance() { }
    public void setBalance(double val) { }
    public String getName() { }
    public void setName(String aName) { }
    public boolean deposit(double amount) { }

    // Same as Slide 8
    public boolean withdraw(double amount) { }
    public void display() { }
}
```
Interfaces or Abstract classes

• Java allows you to implement as many interfaces as you like
  – you can only extend one abstract class not more!

• Abstract classes can also contain state (instance variables) and implemented methods
  – interfaces cannot have instance variables (they can have static variables) and cannot have implementations for methods
Interfaces or Abstract classes (cont)

- Both define a type
  - with abstract classes you can also share implementation and enforce method signatures to be implemented later
  - with interfaces you can only enforce method signatures that need to be implemented later
  - A class can implement multiple interfaces but extend *only* one class (concrete or abstract)
Types Revisited

- In Java each interface defines a type. Interface extension and implementation as subtype relationships

- A subtype relation in Java is:
  - if class $C_1$ extends class $C_2$ then $C_1$ is a subtype of $C_2$
  - if interface $I_1$ extends $I$ then $I_1$ is a subtype of $I$
  - if class $C$ implements interface $I$ then $C$ is a subtype of $I$
  - for every interface $I$, $I$ is a subtype of $\text{Object}$
  - for every type $T$, $T[\ ]$ is a subtype of $\text{Object}$
  - if $T_1$ is a subtype of $T_2$ then $T_1[\ ]$ is a subtype of $T_2[\ ]$
Upcasting

- Operation that changes the runtime type of an instance to one of its supertypes (i.e. move up the hierarchy)
  - force an instance that is of type Savings Account to be viewed as of type Account.

```java
public class Main{

    public static void main(String[] args){
        Savings mySavings = new Savings("John", 100.00, 50.00);
        Savings anotherSavings = new Savings("Mary", 200.00);
        Checking cAccount = new Checking("Michael", 89.00);
        mySavings.display();
        mySavings.display("Today");

        Account oneAC = (Account) mySavings;
        Account secondAC= (Account) cAccount;
        Account[] allAccounts = new Account[10];
        allAccounts[0] = oneAC;
        allAccounts[1] = secondAC;
    }
}
```

Force mySavings to be used as of type Account
Downcasting

- Operation that changes the runtime type of an instance to one of its supertypes (i.e. move down the hierarchy)
  - force an instance that is of type Account to be viewed as of type Savings Account.

```java
public class Main{

public static void main(String[] args){
    Savings mySavings = new Savings("John", 100.00, 50.00);
    Savings anotherSavings = new Savings("Mary", 200.00);
    Checking cAccount = new Checking("Michael", 89.00);
    mySavings.display();
    mySavings.display("Today");

    Account oneAC = (Account) mySavings;
    Account secondAC= (Account) cAccount;
    Account[] allAccounts = new Account[10];
    allAccounts[0] = oneAC;
    allAccounts[1] = secondAC;

    Savings saveAcc = (Savings) allAccounts[0];
    Savings saveAcc2 = (Savings) allAccounts[1];
}
```