Class 09 Slides: Polymorphism

Preconditions

- Students are familiar with inheritance and arrays.
- Students have worked with a poorly written program in A08 that could benefit from polymorphism.
- Students have read Chapter 12 of the text.

Postconditions

- Students have seen polymorphism at work and have seen the results of not using it.

Table of Contents

1 Review Inheritance and Overriding *(20 min)*

2 A Polymorphic Example: Dancing Robots *(30*

2.1 LeftDancers and RightDancers
2.2 Polymorphic Variables
2.3 Using an Array
2.4 Using Methods Unique to a Subclass

3 Designing with Inheritance *(30 min)*

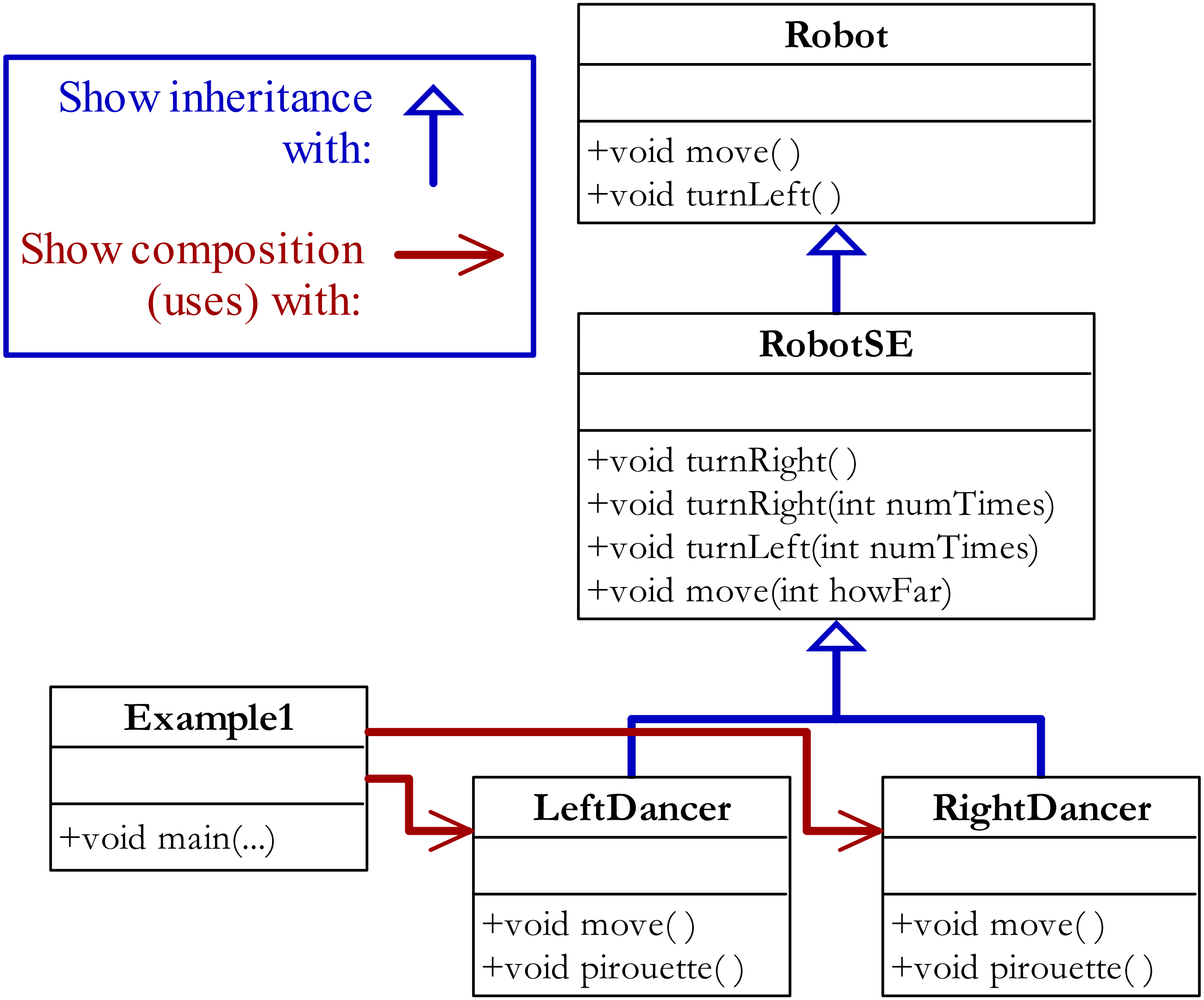
3.1 A Poor Design
3.2 Using Polymorphism
3.2.1 Changes to Withdraw in Bank class
3.2.2 Changes to findAccount in Bank class
3.2.3 Changes to add a new kind of Account

4 Summary

Overriding Move

```
import becker.robots.*;

/** A robot which “dances" towards the left.
@author Byron Weber Becker*/
public class LeftDancer extends RobotSE
{  public LeftDancer(City c, int ave, int str, int dir)
   {  super(c, ave, str, dir);
   }

   public void move()
   {  this.turnLeft();
      super.move();
      this.turnRight();
      super.move();
      this.turnRight();
      super.move();
      this.turnLeft();
   }

   public void pirouette()
   {  this.turnLeft(4);
   }
}
```

Ex1: Program Design



Show inheritance with:

Show composition (uses) with:

Robot

+void move( )
+void turnLeft( )

RobotSE

+void turnRight( )
+void turnRight(int numTimes)
+void turnLeft(int numTimes)
+void move(int howFar)

Example1

+void main(...)

LeftDancer

+void move( )
+void pirouette( )

RightDancer

+void move( )
+void pirouette( )

LeftDancers and RightDancers

```
import becker.robots.*;

/** A robot which "dances" towards the left.
@author Byron Weber Becker*/
public class LeftDancer
          extends RobotSE
{ //constructor omitted for brevity
  public void move()
  { this.turnLeft();
    super.move();
    this.turnRight();
    super.move();
    this.turnRight();
    super.move();
    this.turnLeft();
  }

  public void pirouette()
  { this.turnLeft(4);
  }
}
```

```
import becker.robots.*;

/** A robot which "dances" towards the right.
@author Byron Weber Becker*/
public class RightDancer
          extends RobotSE
{ //constructor omitted for brevity
  public void move()
  { this.turnRight();
    super.move();
    this.turnLeft();
    super.move();
    this.turnLeft();
    super.move();
    this.turnRight();
  }

  public void pirouette()
  { this.turnRight(4);
  }
}
```

Ex1: Dancing Robots

```
import becker.robots.*;

public class Example1 extends Object
{  public static void main(String[ ] args)
   {  City danceFloor = new City();
      LeftDancer ld = new LeftDancer(danceFloor, 1, 4, Directions.NORTH);
      RightDancer rd = new RightDancer(danceFloor, 2, 4, Directions.NORTH);
      CityFrame f = new CityFrame(danceFloor, 4, 5);

      for (int i=0; i< 4; i++)
      {  ld.move();
         rd.move();
      }

      ld.pirouette();
      rd.pirouette();
   }
}
```

Ex. 2: Polymorphic Variables

```
import becker.robots.*;

public class Example2 extends Object
{  public static void main(String[ ] args)
   {  City danceFloor = new City();

      Robot ld = new LeftDancer(danceFloor, 1, 4, Directions.NORTH);
      Robot rd = new RightDancer(danceFloor, 2, 4, Directions.NORTH);
      CityFrame f = new CityFrame(danceFloor, 4, 5);

      for (int i=0; i< 4; i++)
      {  ld.move();
         rd.move();
      }

      ld.pirouette();
      rd.pirouette();
   }
}
```

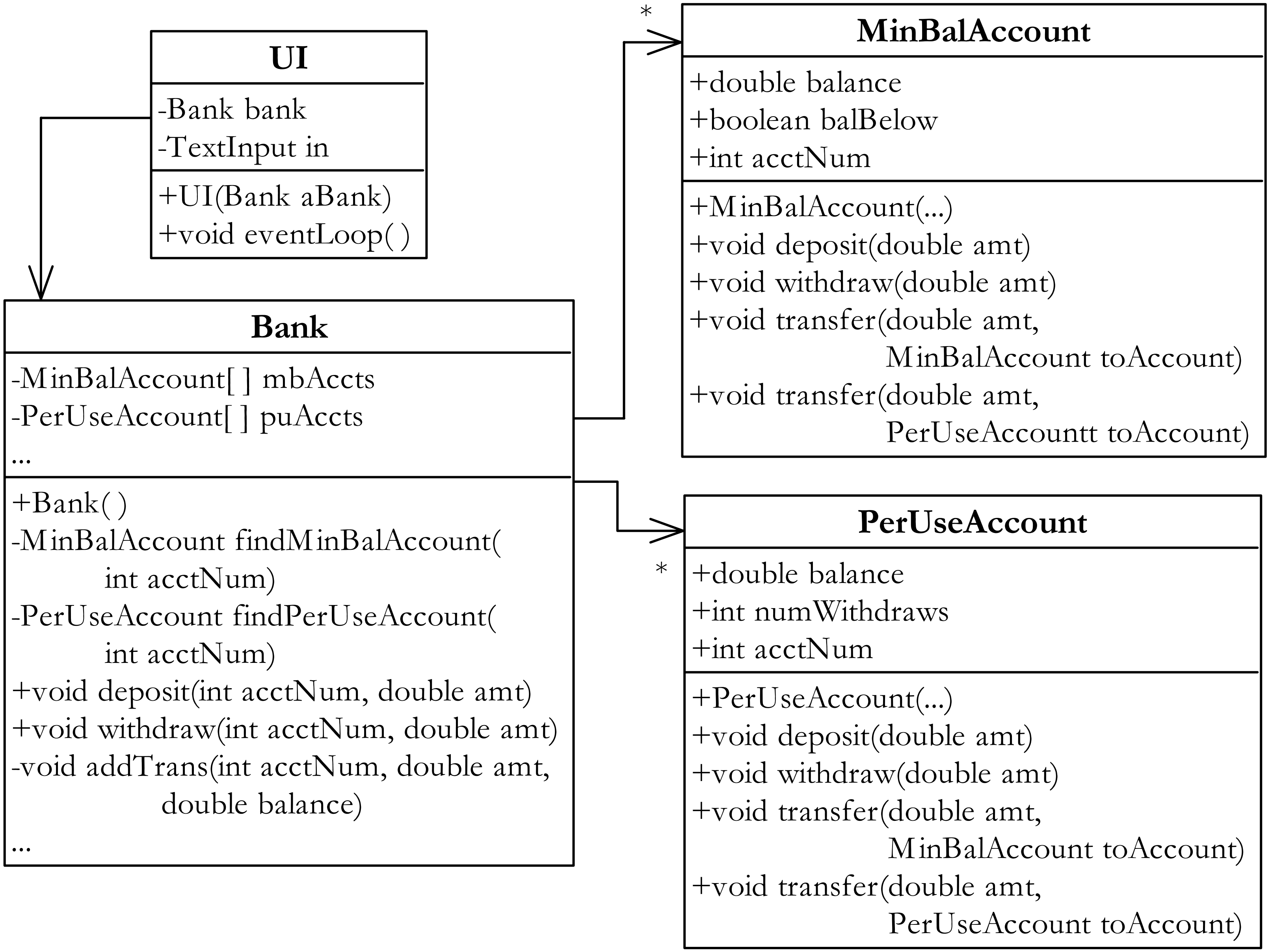
Ex. 3: Using an Array

```
public class Example3 extends Object
{  public static void main(String args[ ])
   {  City danceFloor = new City();

      Robot[ ] chorusLine = new Robot[4];
      for(int i=0; i<chorusLine.length; i++)
      {  if (i%3 == 0)
            chorusLine[i] = new LeftDancer(danceFloor, 1+i, 4, Directions.NORTH);
         else if (i%3 == 1)
            chorusLine[i] = new RightDancer(danceFloor, 1+i, 4, Directions.NORTH);
         else
            chorusLine[i] = new Robot(danceFloor, 1+i, 4, Directions.NORTH);
      }

      for(int i=0; i<4; i++)
      {  for(int j=0; j<chorusLine.length; j++)
         {  chorusLine[j].move();
         }
      }

   }
}
```

Ex. 4: Methods Unique to a Subclass

```
    // Identical to Example 3 up to here.

    // Make them dance
    for (int i=0; i<4; i++)
    {  for(int j=0; i< chorusLine.length; i++)
       {  chorusLine[j].move();
       }
    }

    // End with a pirouette (if able)
    for(int i=0; i<chorusLine.length; i++)
    {









    }
  }
}
```

Accounts: A Poor Design



UI

-Bank bank
-TextInput in

+UI(Bank aBank)
+void eventLoop( )

Bank

-MinBalAccount[ ] mbAccts
-PerUseAccount[ ] puAccts
...

+Bank( )
-MinBalAccount findMinBalAccount(
int acctNum)
-PerUseAccount findPerUseAccount(
int acctNum)
+void deposit(int acctNum, double amt)
+void withdraw(int acctNum, double amt)
-void addTrans(int acctNum, double amt,
double balance)
...

*

MinBalAccount

+double balance
+boolean balBelow
+int acctNum

+MinBalAccount(...)
+void deposit(double amt)
+void withdraw(double amt)
+void transfer(double amt,
MinBalAccount toAccount)
+void transfer(double amt,
PerUseAccountt toAccount)

*

PerUseAccount

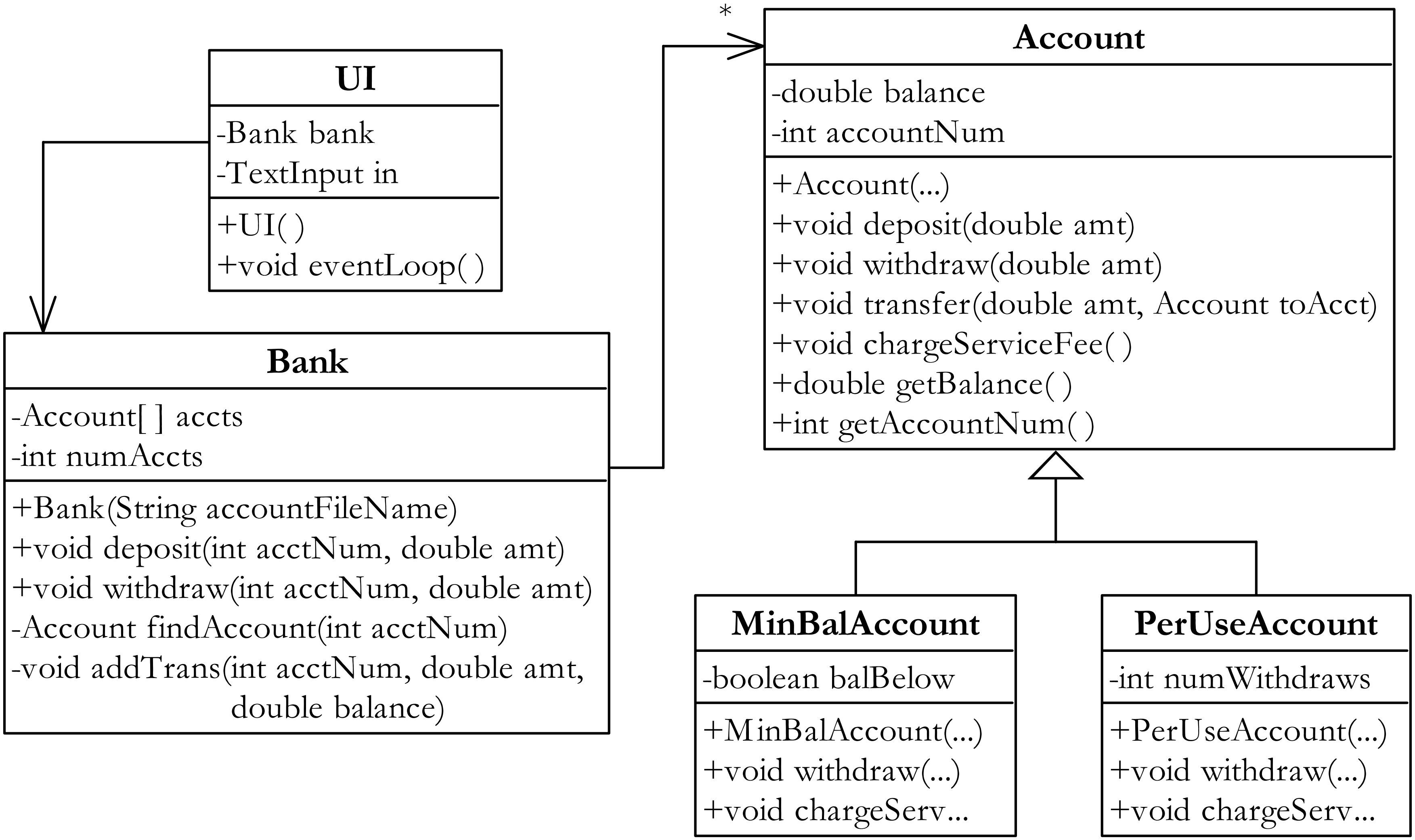
+double balance
+int numWithdraws
+int acctNum

+PerUseAccount(...)
+void deposit(double amt)
+void withdraw(double amt)
+void transfer(double amt,
MinBalAccount toAccount)
+void transfer(double amt,
PerUseAccount toAccount)

Accounts: Code From A Poor Design

```
public class Bank extends Object
{  private MinBalAccount[ ] mbAccts;
   private PerUseAccount[ ] puAccts;
   …
   public void deposit(int acctNum, double amt)
   {  // Look for this account in the list of min balance accounts. If there, do the deposit.
      MinBalAccount mba = this.findMinBalAccount(acctNum);
      if (mba != null)
      {  mba.deposit(amt);
         this.addTrans(acctNum, amt, mba.balance);
      } else
      {  /* Wasn't in the min balance accounts list. Look in the per-use accounts list. If
         there, do the deposit. */
         PerUseAccount pua = this.findPerUseAccount(acctNum);
         if (pua != null)
         {  pua.deposit(amt);
            this.addTrans(acctNum, amt, pua.balance);
         } else
         {  System.out.println("Account " + acctNum + " not found.");
         }
      }
   }
```

A Design Using Polymorphism



UI

-Bank bank
-TextInput in

+UI( )
+void eventLoop( )

Bank

-Account[ ] accts
-int numAccts

+Bank(String accountFileName)
+void deposit(int acctNum, double amt)
+void withdraw(int acctNum, double amt)
-Account findAccount(int acctNum)
-void addTrans(int acctNum, double amt, double balance)

Account

-double balance
-int accountNum

+Account(...)
+void deposit(double amt)
+void withdraw(double amt)
+void transfer(double amt, Account toAcct)
+void chargeServiceFee( )
+double getBalance( )
+int getAccountNum( )

MinBalAccount

-boolean balBelow

+MinBalAccount(...)
+void withdraw(...)
+void chargeServ...

PerUseAccount

-int numWithdraws

+PerUseAccount(...)
+void withdraw(...)
+void chargeServ...

Code Using Polymorphism

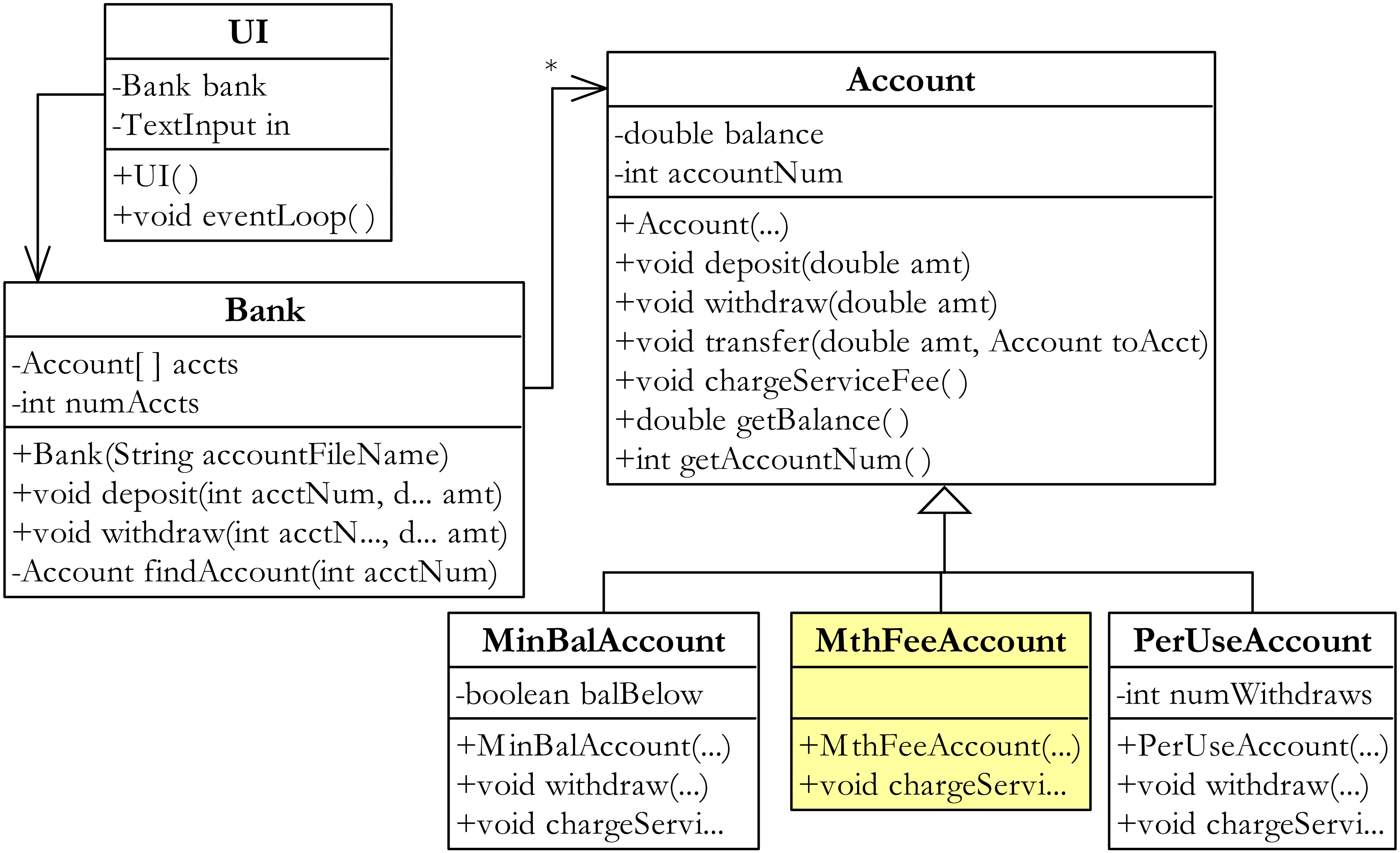
```
public class Bank extends Object
{  private Account[ ] accts;
   private int numAccts;
   …
   public void withdraw(int acctNum, double amt)
   {
```

More Code Using Polymorphism

```
public class Bank extends Object
{  private Account[ ] accts;
   private int numAccts;
   …

   private Account findAccount(int acctNum)
   {  int i = 0;
      while (true)
      {  if (i >= this.numAccts)
         {  return null;
         } else if (this.accts[i].getAccountNum() == acctNum)
         {  return this.accts[i];
         } else
         {  i++;
         }
      }
   }

   …
}
```

Adding MthFeeAcct

What changes are needed in **Bank** to add a monthly fee account?



Summary

Polymorphism…

- is when objects respond to the same message (method name) in different ways, depending on their type.
- is implemented by extending a class with two or more subclasses. The methods in the superclass may be overridden by subclasses to respond differently.
- can substantially simplify programs, making them easier to read, write, understand, test, debug, and change.