CHAPTER 10

INHERITANCE

Inheritance

Inheritance: extend classes by adding or redefining methods, and adding instance fields

Example: Savings account = bank account with interest

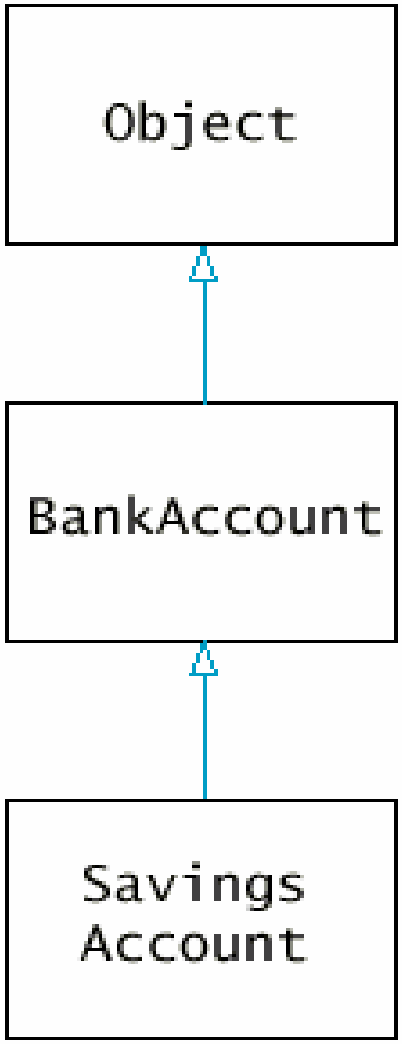
```
class SavingsAccount extends BankAccount
{
    new methods
    new instance fields
}
```

All methods of `BankAccount` are automatically inherited

Ok to call `deposit, getBalance` on `SavingsAccount` object

Extended class = *superclass*, extending class = *subclass*

An Inheritance Diagram



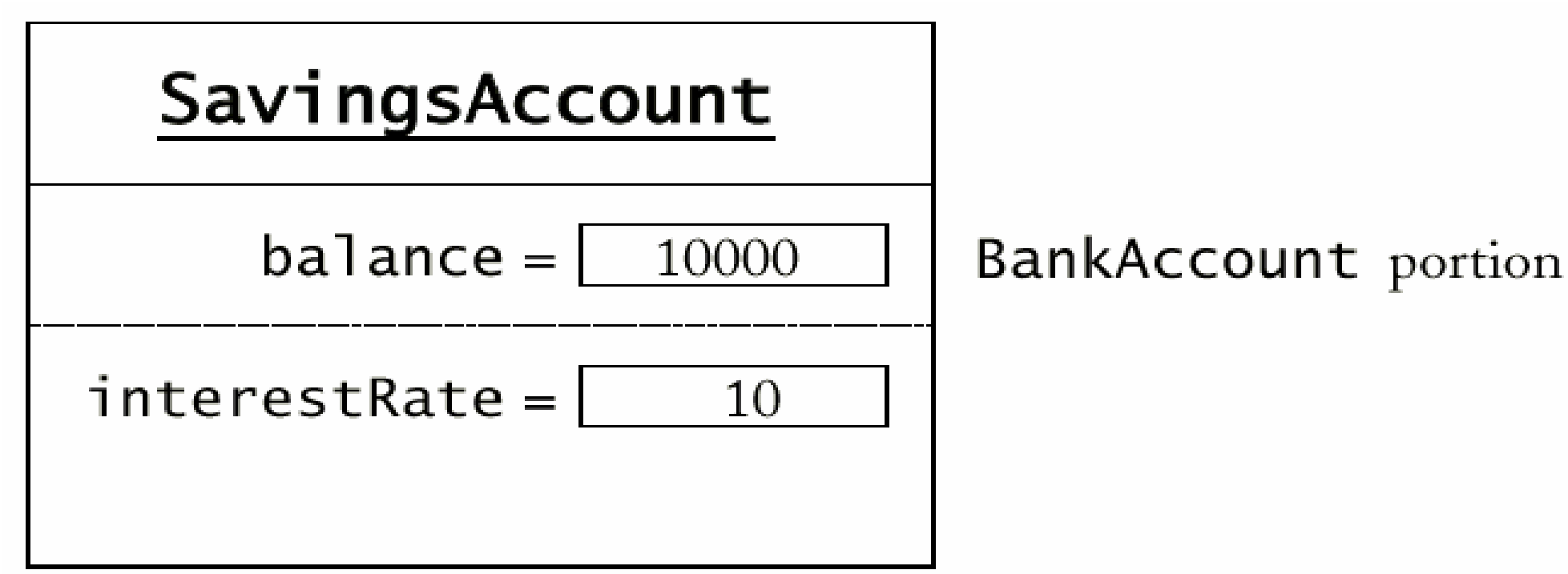
Adding a Subclass Method

```
public class SavingsAccount extends BankAccount
{
  public SavingsAccount(double rate)
  {
        interestRate = rate;
  }

  public void addInterest()
  {
        double interest = getBalance()*interestRate/100;
        deposit(interest);
  }

  private double interestRate;
}
```

Layout of a Subclass Object



Syntax 10.1: Inheritance

```
class SubclassName extends SuperclassName
{
   methods
   instance fields
}
```

Example:

```
public class SavingsAccount extends BankAccount
{
  public SavingsAccount(double rate)
  {
     interestRate = rate;
  }
  public void addInterest()
  {
     double interest = getBalance()*interestRate/100;
     deposit(interest);
  }
  private double interestRate;
}
```

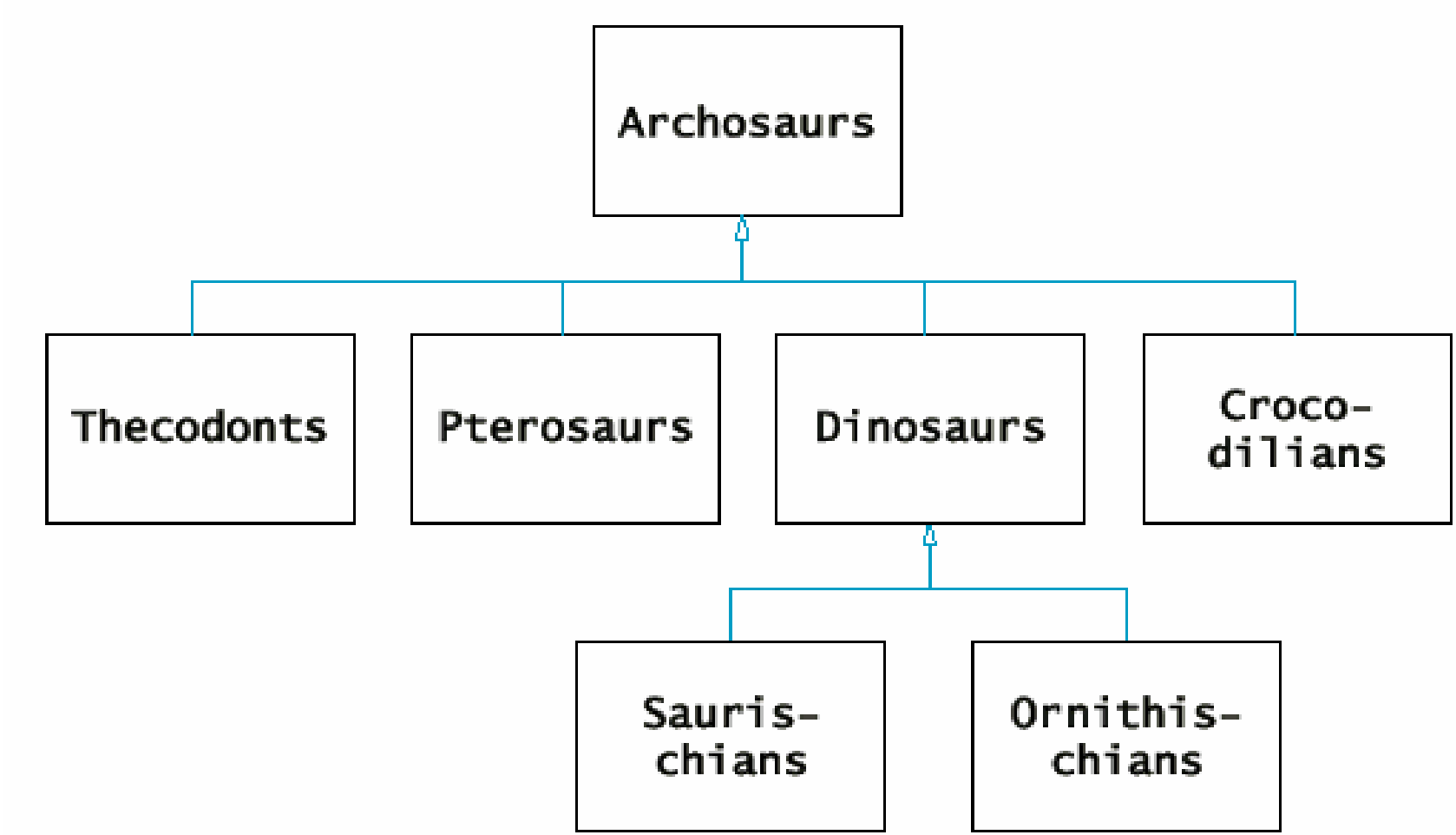
Purpose:

To define a new class that inherits from an existing class, and define the methods and instance fields that are added in the new class.

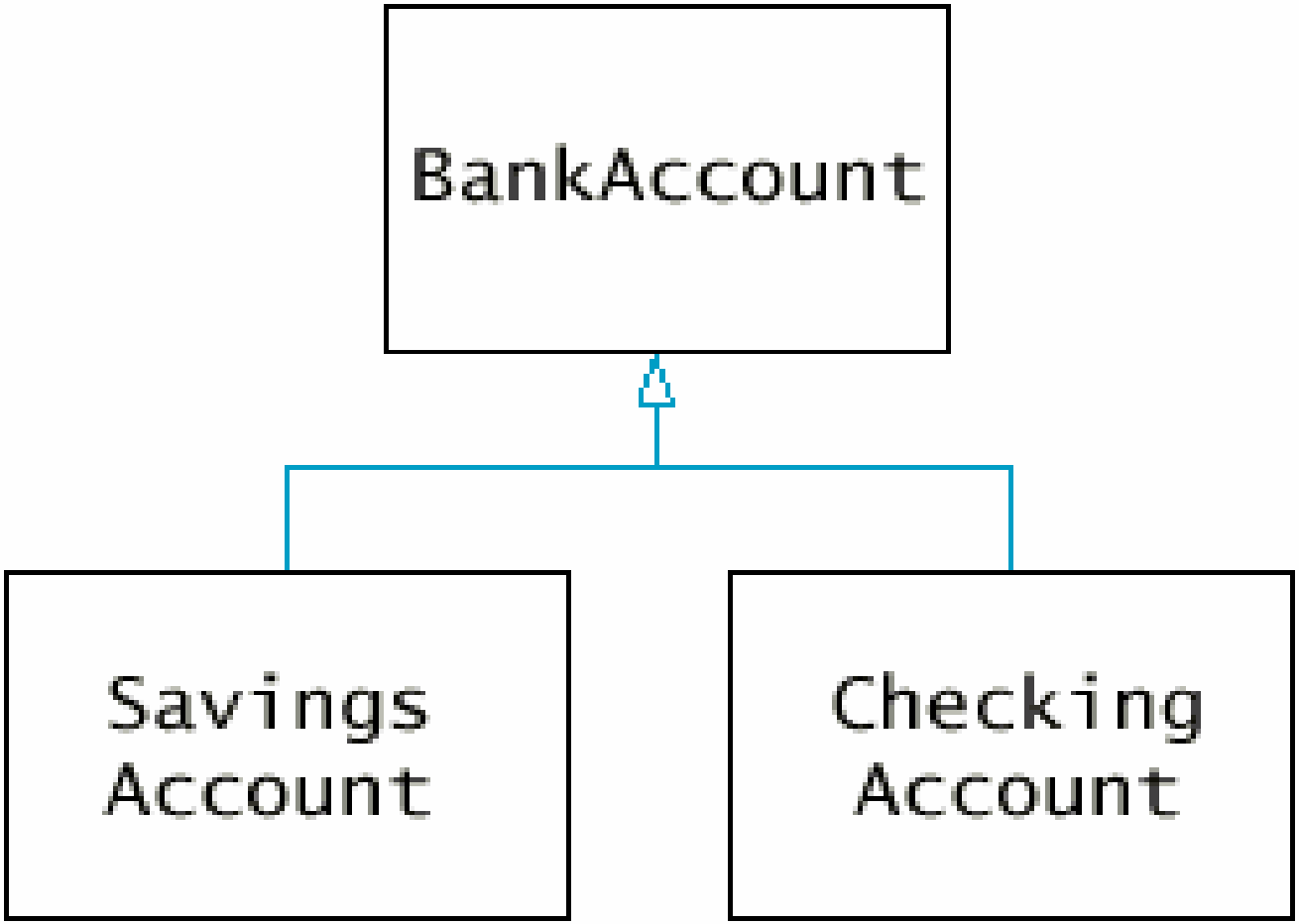
Inheritance Hierarchies

- Hierarchies of classes, subclasses, and sub-subclasses are common
- Real world example: Ancient reptiles
- We will study a simple bank account hierarchy

A Part of the Hierarchy of Ancient Reptiles



Bank Account Hierarchy



Inheritance and Methods

- Override method: Supply a different implementation of a method that exists in the superclass
- Inherit method: Don't supply a new implementation of a method that exists in the superclass
- Add method: Supply a new method that doesn't exist in the superclass

Inheritance and Fields

- Inherit field: All fields from the superclass are automatically inherited
- Add field: Supply a new field that doesn't exist in the superclass
- Can't override fields (rather the subclass field shadows the superclass field)

CheckingAccount Class

- First three transactions are free
- Charge $2 for every additional transaction
- Must override `deposit` and `withdraw` to increment transaction count
- The method `deductFees` deducts accumulated fees, resets transaction count

Inherited Fields are Private

- Consider `deposit` method of `CheckingAccount`

```
public void deposit(double amount)
{
   transactionCount++;
   // now add amount to balance
   ...
}
```

- Can't just add `amount` to `balance`
- The field `balance` is a *private* field of the superclass
- Subclass must use public interface

Invoking a Superclass Method

- Can't just call
  `deposit(amount)`
  in `deposit` method of `CheckingAccount`
- Calls the same method (infinite recursion)
- Instead, invoke *superclass method*
  `super.deposit(amount)`
- Now calls `deposit` method of `BankAccount` class
- Complete method:

```
public void deposit(double amount)
{
   transactionCount++;
   super.deposit(amount);
}
```

Syntax 10.2: Calling a Superclass Method

`super.`*methodName*`(`*parameters*`)`

Example:

```
public void deposit(double amount)
{
  transactionCount++;
  super.deposit(amount);
}
```

Purpose:

To call a method of the superclass instead of the method of them current class

Superclass Construction

```
public class CheckingAccount extends BankAccount
{
  public CheckingAccount(double initialBalance)
  {
      // construct superclass
      super(initialBalance);
      // initialize transaction count
      transactionCount = 0;
  }
  ...
}
```

Syntax 10.3: Calling a Superclass Constructor

```
ClassName(parameters)
{
  super(parameters);
  . . .
}
```

Example:

```
public CheckingAccount(double initialBalance)
{
  super(initialBalance);
  transactionCount = 0;
}
```

Purpose:

To invoke a constructor of the superclass. Note that this statement must be the first statement of the subclass constructor.

Converting from Subclasses to Superclasses

- Ok to convert subclass reference to superclass reference

```
SavingsAccount collegeFund =
new SavingsAccount(10);
BankAccount anAccount = collegeFund;
Object anObject = collegeFund;
```

- Superclass references don't know the full story:

```
anAccount.addInterest(); // ERROR
```

- Why would anyone want to know *less* about an object?

Polymorphism

- Generic method:

```
public void transfer(double amount, BankAccount other)
{
   withdraw(amount);
   other.deposit(amount);
}
```

- Works with any kind of bank account (plain, checking, savings)
- Note polymorphism:
  `other.deposit(amount)`
  calls `CheckingAccount.deposit` (and charges transaction fee)
  if `other` refers to a checking account
- Why not just declare parameter as `Object`?
- The `Object` class doesn't have `deposit` method

File AccountTest.java

```
public class AccountTest
{
   public static void main(String[] args)
   {
    SavingsAccount momsSavings =
    new SavingsAccount(0.5);
    CheckingAccount harrysChecking =
    new CheckingAccount(100);
    momsSavings.deposit(10000);
    momsSavings.transfer(2000, harrysChecking);
    harrysChecking.withdraw(1500);
    harrysChecking.withdraw(80);
    momsSavings.transfer(1000, harrysChecking);
    harrysChecking.withdraw(400); // simulate end of month
    momsSavings.addInterest();
    harrysChecking.deductFees();
    System.out.println("Mom's savings balance = $" +
    momsSavings.getBalance());
    System.out.println("Harry's checking balance = $" +
    harrysChecking.getBalance());
   }
}
```

File BankAccount.java

```
public class BankAccount
{
   public BankAccount()
   {
    balance = 0;
   }

   public BankAccount(double initialBalance)
   {
    balance = initialBalance;
   }

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

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

   public double getBalance()
   {
    return balance;
   }

   public void transfer(double amount, BankAccount other)
   {
    withdraw(amount);
    other.deposit(amount);
   }

   private double balance;
}
```

File CheckingAccount.java

```
public class CheckingAccount extends BankAccount
{
   public CheckingAccount(int initialBalance)
   {
    super(initialBalance);
    transactionCount = 0;
   }
   public void deposit(double amount)
   {
    transactionCount++;
    super.deposit(amount);
   }
   public void withdraw(double amount)
   {
    transactionCount++;
    super.withdraw(amount);
   }
```

```
   public void deductFees()
   {
    if (transactionCount > FREE_TRANSACTIONS)
    {
          double fees = TRANSACTION_FEE *
          (transactionCount - FREE_TRANSACTIONS);
          super.withdraw(fees);
    }
    transactionCount = 0;
   }

   private int transactionCount;
   private static final int FREE_TRANSACTIONS = 3;
   private static final double TRANSACTION_FEE = 2.0;
}
```

File SavingsAccount.java

```
public class SavingsAccount extends BankAccount
{
   public SavingsAccount(double rate)
   {
    interestRate = rate;
   }

   public void addInterest()
   {
    double interest = getBalance()*interestRate/100;
    deposit(interest);
   }

   private double interestRate;
}
```