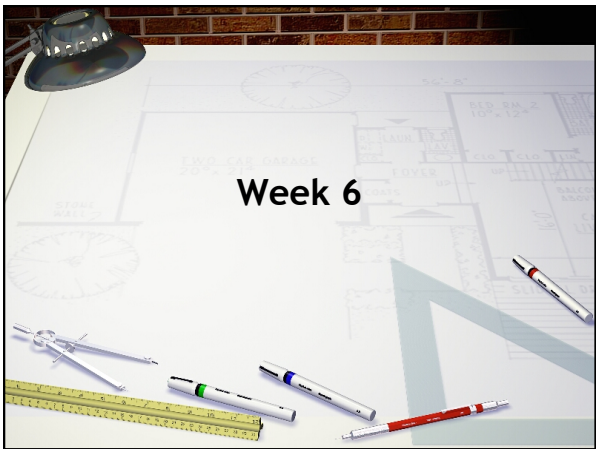
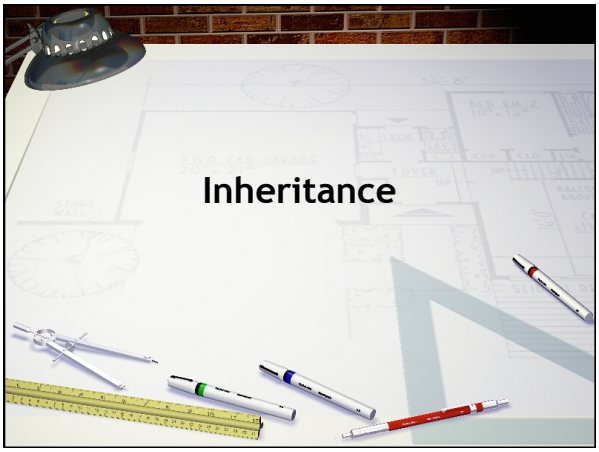

Week 6


Mid Course Assignment


Inheritance

Inheritance

- One of the most powerful tools in object-oriented programming.
- Java provides two forms:
  - Class inheritance
    - Referred to as class extension
  - Interface inheritance

Inheritance
Class Extension

- Extending an existing class to create another class.
- When a class extends another it "inherits" all behavior and attributes of the class it is extended from.
- The syntax for class extension is:

```
[modifiers] class className [extends baseName]
{ body }
```

Inheritance
Class Extension (cont.)

- Single Inheritance
  - A class may extend exactly one class
  - If not explicitly extended, extends `Object`
- Allows additional methods and variables to be added to the new class while preserving behavior and data inherited from the superclass.

Inheritance
Object class

- Methods on class `Object`

```
Class getClass()
boolean equals( Object obj )
int hashCode()
String toString()
```

Javadoc - http://java.sun.com/javase/6/docs/api/java/lang/Object.html

Inheritance
Class class

- Methods on class `Class`

```
String getName()
static Class forName(String clsName)
Object newInstance()
```

JavaDoc - http://java.sun.com/javase/6/docs/api/java/lang/Class.html

Inheritance
Method Overriding

- A form of polymorphism
  - Made possible by class extension
- Entails re-implementing a method from the superclass in the subclass
- Behavior for a specific method may be altered for a single subclass

Inheritance
Abstract Classes

- Useful where part of the implementation can be define
- And, no valid default implementation available for some operations
- Cannot be instantiated

Interface

Inheritance

- The syntax for an interface:

```
[modifiers] interface interface_name
[extends interface_1, interface_2, … interface_n]
{ body }
```

- May extend (inherit) from any number of other interfaces.
- The *body* of an interface consists of
  - Variable declarations, must be static
  - Method declarations
- Provide no behavior

Inheritance
Interface Usage

- Classes may implement an interface
  - The class provides an implementation of the methods declared in the interface
- Interfaces are particularly appropriate were the implementation of a method varies across all the implementing classes

Inheritance
Interface Implementation

- The syntax for class declaration is extended to specify interfaces implemented:

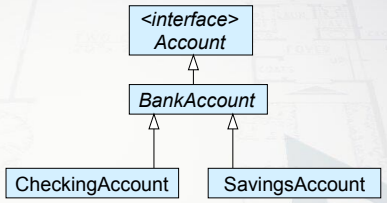
```
[modifiers] class className [extends baseName]
[implements interface_1, … interface_n]
{ body }
```

- A class may implement any number of interfaces

Inheritance
Example

- `Account`
  - Defines interface required of all accounts
- `BankAccount`
  - Serves as a superclass for checking and savings account classes, implements basic methods
- `SavingsAccount`
  - Inherits its entire behavior from `BankAccount`
- `CheckingAccount`
  - Adds an `overDraftAccount` attribute
  - Adds a `setOverDraftAccount()` method
  - Overrides the `withdraw()` method

Inheritance Hierarchies



Bank Account Example

Account Interface

```
public interface Account {
    void deposit(double amount);
    boolean withdraw(double amount);
    double getBalance();
    double getInterestRate();
    void payDividend();
}
```

BankAccount Class

```
public abstract class BankAccount implements Account {
    protected double balance, interestRate;
    public void deposit(double amount) {
        balance += amount;
    }
    public double getBalance() {
        return balance;
    }
   public double getInterestRate() {
        return interestRate;
   }
   public void payDividend() {
        balance *= (1 + interestRate);
   }
   public abstract boolean withdraw(double amount);
}
```

SavingsAccount Class

```
public class SavingsAccount extends BankAccount {

        public boolean withdraw(double amount) {
            boolean status = true;

            if(balance >= amount) {
                 balance -= amount;
            } else {
                 status = false;
            }
            return status;
         }
}
```

CheckingAccount Class

```
public class CheckingAccount extends BankAccount {
    private BankAccount overDraftAccount;
    public boolean withdraw(double amount) {
         boolean status = true;
         if (balance >= amount) {
             balance -= amount;
         } else if (overDraftAccount != null &&
              overDraftAccount.getBalance() >= amount-balance) {
             status = overDraftAccount.withdraw(amount-balance);
             balance = 0;
         } else {
              status = false;
         }
          return status;
    }
```

CheckingAccount Class

```
    public void setOverDraftAccount(BankAccount account) {
        overDraftAccount = account;
    }
}
```

Super constructor/Casting

Superclass Constructor

- Subclass constructors must always contain a 'super' call
- If none is written, the compiler inserts one (without parameters)
  - works only, if the superclass has a constructor without parameters
- Must be the first statement in the subclass constructor

Using super

- Methods of subclasses may use the `super` keyword to invoke overridden methods of the superclass

```
...
public String toString() {
    String s = super.toString();
    s += ...;
    return s;
}
...
```

Subclasses and Subtyping

- Classes define types
- Subclasses define subtypes
- Substitution - objects of subclasses can be used where objects of supertypes are required
- Object variables in Java are **polymorphic**
  (They can hold objects of more than one type.)
  - They can hold objects of the declared type, or of subtypes of the declared type
  - Subclass objects may be assigned to superclass variables
  - Subclass objects may be passed to superclass parameters

Casting Revisited

- Can assign subtype to supertype.
- Cannot assign supertype to subtype!

```
Account acct = ...;
CheckingAccount chkAcct = acct;   error!
```

- Casting fixes this:

```
CheckingAccount chkAcct =
                (CheckingAccount)acct;
```

  - Only if the `acct` really is a `CheckingAccount`, otherwise `ClassCastException`

Using `instanceof`

- The `instanceof` operator tests if an object extends/implements a specific class/ interface
- Allow you to ensure that a cast will succeed

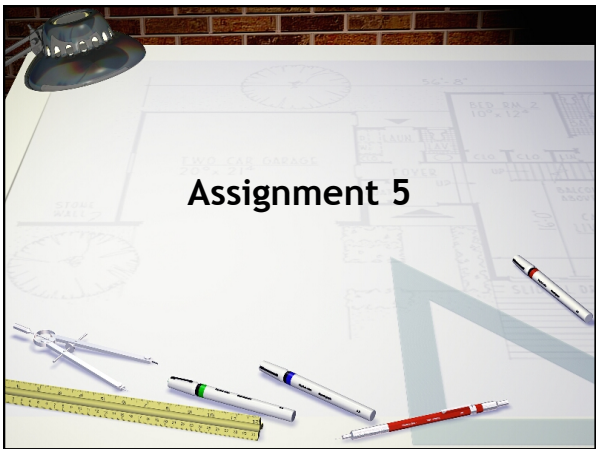
```
...
if (acct instanceof CheckingAccount) {
    CheckingAccount chkAcct = (CheckingAccount)acct;
    ...
}
...
```

Polymorphic Collections

- All collections are polymorphic
- The elements are of type Object
- Generics let us limit polymorphic types

Preventing Polymorphism

- Methods declared as `final` can't be overridden
- Classes declared as `final` can't be extended


Assignment 5