Inheritance
Abstract Classes

Inheritance

- Object oriented languages have a feature called **inheritance**
- Inheritance enables you to define a new class, based upon an existing class
- The new class is similar to the existing class, but it has additional member variables and methods
- This makes programming easier, because you can build upon an existing class, instead of starting out from scratch
- Programming in Java consists mostly of creating class hierarchies and instantiating objects from them

Exercise 1

- Create a class, called **Video**, to represent videos available at a rental store
- The class **Video** has three *private* member variables:
  - *String title;* //name of the item
  - *int length;* //number of minutes
  - *boolean available;* //is the video in the store?

Exercise 1

- There are two constructors in the class
- The first one has only one parameter, the title of the video, and initializes the other member variables of the class with the following values: *length = 90* and *available = true*
- The second constructor has two parameters, the title of the video and its length, while the member variable *available* is initialized to *true*
- The method *show()* displays information regarding the video objects

Exercise 1

- In another class, called **VideoTest**, which contains the *main* method, create two objects of class **Video**
- The first object is created using the first constructor of the class, and the second object is created using the second constructor of the class
- Display on the screen the information regarding the video objects

Exercise 2

- Create a class, called **Movie**, which inherits the class **Video**, and has, in addition, two member variables: the *director* of a movie and the *rating* of a movie
- The class **Movie** is a subclass of **Video**
- The class **Movie** has a constructor that initializes the data of **Movie** objects
- The method *show()* displays information regarding the movie objects

Observations

- Use the keyword *super* to invoke the constructor of the parent class to initialize some of the data
- *super(…)* must be the first statement in the subclass's constructor
- A constructor for a children class always starts with an invocation of one of the constructors in the parent class
- If the parent class has several constructors, then the one which is invoked is determined by matching argument lists
- Even though the parent class has a *show()* method, the new definition of *show()* in the children class will **override** the parent's version
- A children's method **overrides** a parent's method when it has the same signature as a parent method

Exercise 2

- In another class, called **MovieTest**, which contains the *main* method, create an object of class **Video** and an object of class **Movie**
- Display on the screen the information regarding the two objects

Abstract Classes

- An **abstract class** in Java is a class that is never instantiated
- Its purpose is to be a parent to several related classes
- The children classes inherit from the abstract parent class
- The advantage of using an abstract class is that you can group several related classes together as siblings
- Grouping classes together is important in keeping a program organized and understandable
- Access modifiers such as *public* can be placed before *abstract*
- Even though it can not be instantiated, an abstract class can define methods and variables that children class inherit

Exercise 3

- Create an abstract class, called **Card**, which contains:
- a) the *protected* member variable of type *String* called *recipient* (representing the name of the person who gets the card)
- b) *public abstract void greeting();*
- Each class has its own version of the *greeting()* method
- Each class has a *greeting()*, but each one is implemented differently
- It is useful to put an abstract *greeting()* method in the parent class
- This says that each children inherits the “idea” of *greeting()*, but each implementation is different

Observations

- Since no constructor is defined in **Card**, the default no argument constructor is automatically supplied by the compiler
- However, this constructor cannot be used directly, because no **Card** object can be constructed
- Abstract classes are used to organize the “concept” of something that has several different versions in the children classes
- The abstract class can include abstract methods and non-abstract methods

Exercise 3

- Create a class, called **Holiday**, which is a non-abstract children of an abstract parent class
- The constructor of the class **Holiday** initializes the name of the *recipient* with the parameter received as argument
- The method body for *greeting()* is:
- *System.out.println(“Dear ” + recipient + “, “);*
- *System.out.println(“Season’s Greetings!”);*

Exercise 3

- Create a class, called **Birthday**, which is a non-abstract children of an abstract parent class
- It contains the *private* member variable *age*, of type *int*
- The constructor of the class **Birthday** initializes the name of the *recipient* and the *age* with the parameters received as arguments
- The method body for *greeting()* is:
- *System.out.println(“Dear ” + recipient + “, “);*
- *System.out.println(“Happy ” + age + “th Birthday!”);*

Exercise 3

```
public class CardTest {
public static void main(String[] args) {
Card card1 = new Holiday("John");
card1.greeting();
Card card2 = new Birthday("Betty", 18);
card2.greeting();
}
}
Dear John,
Season's Greetings!
Dear Betty,
Happy 18th Birthday!
```