Using Inheritance

• This chapter emphasizes software design using inheritance

• We have seen inheritance before, such as LinkedList and ArrayList both implementing List

• We go through a sequence of examples
  – A student and a staff member are two examples of a person at a university
  – A CD and a Video are both examples of a multimedia component
  – All Java classes are subclasses of the Object class
A First Example

• This example is not covered in the textbook explicitly, but is available in the exercises associated with the textbook

• We will look at this example of a university database that includes students and staff

• We will expand this database to include an address class that is used by all people associated with the university

• This example will pave the way to examine the DoME example in the textbook
The Class Structures
import java.util.ArrayList;
import java.util.Iterator;

/**
 * A very simple database of people in a university. This class simply stores
 * persons and, at request, lists them on standard output.
 * @author Michael Kolling
 * @version 1.1, March 2002
 */

public class Database {
    private ArrayList persons;

    public Database()
    {
        persons = new ArrayList();
    }

    public void addPerson(Person p)
    {
        persons.add(p);
    }

    public void listAll ()
    {
        for (Iterator i = persons.iterator(); i.hasNext();)
        {
            System.out.println(i.next());
        }
    }
}
/**
* A person class for a simple BlueJ demo program. Person is used as
* an abstract superclass of more specific person classes.
* 
* @author  Michael Kolling
* @version 1.0, January 1999
*/

abstract class Person
{
    private String name;
    private int yearOfBirth;

    /**
     * Create a person with given name and age.
     */
    Person(String name, int yearOfBirth)
    {
        this.name = name;
        this.yearOfBirth = yearOfBirth;
    }

    /**
     * Set a new name for this person.
     */
    public void setName(String newName)
    {
        name = newName;
    }
}
public String getName()
{
    return name;
}

public void setYearOfBirth(int newYearOfBirth)
{
    yearOfBirth = newYearOfBirth;
}

/**
 * Return the birth year of this person.
 */
public int getYearOfBirth()
{
    return yearOfBirth;
}

/**
 * Return a string representation of this object.
 */
public String toString()    // redefined from "Object"
{
    return "Name: " + name + "\n" +
           "Year of birth: " + yearOfBirth + "\n";
}

}
class Staff extends Person {
    private String room;

    /**
     * Create a staff member with default settings for detail information.
     */
    Staff() {
        super("(unknown name)", 0000);
        room = "(unknown room)";
    }

    /**
     * Create a staff member with given name, year of birth and room number.
     */
    Staff(String name, int yearOfBirth, String roomNumber) {
        super(name, yearOfBirth);
        room = roomNumber;
    }
}
/**
 * Set a new room number for this person.
 */
public void setRoom(String newRoom)
{
    room = newRoom;
}

/**
 * Return the room number of this person.
 */
public String getRoom()
{
    return room;
}

/**
 * Return a string representation of this object.
 */
public String toString()    // redefined from "Person"
{
    return super.toString() +
            "Staff member\n" +
            "Room: " + room + "\n";
}
/**
 * A class representing students for a simple BlueJ demo program.
 * @author Michael Kolling
 * @version 1.0, January 1999
 */

class Student extends Person {
    private String SID;  // student ID number

    /**
     * Create a student with default settings for detail information.
     */
    Student()
    {
        super("(unknown name)", 0000);
        SID = "(unknown ID)";
    }

    /**
     * Create a student with given name, year of birth and student ID
     */
    Student(String name, int yearOfBirth, String studentID)
    {
        super(name, yearOfBirth);
        SID = studentID;
    }
}
/**
 * Return the student ID of this student.
 */
public String getStudentID()
{
    return SID;
}

/**
 * Return a string representation of this object.
 */
public String toString()     // redefined from "Person"
{
    return super.toString() +
            "Student\n" +
            "Student ID: " + SID + "\n";
}
}
Adding Shared Information

• Both students and staff have local address information

• There are basically three choices to add this information
  – Add it to each of the individual classes
  – Add it to the person class
  – Add it as a separate class and link it to the person class

• Which choice do you think is best?

• Be prepared to defend your choice
public class Address
{
    private String street;
    private String town;
    private String postCode;
    private String country;

    public Address(String street, String town, String postCode)
    {
        this(street, town, postCode, "");
    }

    public Address(String street, String town, String postCode, String country)
    {
        this.street = street;
        this.town = town;
        this.postCode = postCode;
        this.country = country;
    }

    public String toString()
    {
        return street + "\n" +
            town + " " + postCode + "\n" +
            country + "\n";
    }
}
An Introduction to DoME

• DoME stands for Database for Multimedia Entertainment
• Initially DoME will deal with Compact Disks (CD) and video
• The first version does not use inheritance
• After examining this version, you will be asked where inheritance can help create a second version of DoME
• Hopefully our previous example will provide some guidance
import java.util.ArrayList;
import java.util.Iterator;

/**
 * The database class provides a facility to store CD and video
 * objects. A list of all CDs and videos can be printed to the
 * terminal.
 *
 * @author Michael Kolling and David J. Barnes
 * @version 2002-05-02
 */
public class Database
{
    private ArrayList cds;
    private ArrayList videos;

    /**
     * Construct an empty Database.
     */
    public Database()
    {
        cds = new ArrayList();
        videos = new ArrayList();
    }
    // methods go here
}
public void addCD(CD theCD)
{
    cds.add(theCD);
}

public void addVideo(Video theVideo)
{
    videos.add(theVideo);
}

public void list()
{
    // print list of CDs
    for(Iterator iter = cds.iterator(); iter.hasNext(); ) {
        CD cd = (CD)iter.next();
        cd.print();
        System.out.println();   // empty line between items
    }

    // print list of videos
    for(Iterator iter = videos.iterator(); iter.hasNext(); ) {
        Video video = (Video)iter.next();
        video.print();
        System.out.println();   // empty line between items
    }
}
/**
 * The CD class represents a CD object. Information about the
 * CD is stored and can be retrieved.
 *
 * @author Michael Kolling and David J. Barnes
 * @version 2002-05-02
 */

public class CD {

    private String title;
    private String artist;
    private int numberOfTracks;
    private int playingTime;
    private boolean gotIt;
    private String comment;

    public CD(String theTitle, String theArtist, int tracks, int time) {
        title = theTitle;
        artist = theArtist;
        numberOfTracks = tracks;
        playingTime = time;
        gotIt = false;
        comment = "<no comment>";
    }

    // methods go here
}

CD - 1
public void setComment(String comment)
{
    this.comment = comment;
}

public String getComment()
{
    return comment;
}

public void setOwn(boolean ownIt)
{
    gotIt = ownIt;
}

public boolean getOwn()
{
    return gotIt;
}

public void print()
{
    System.out.print("CD: " + title + " (" + playingTime + " mins)");
    if(gotIt) {
        System.out.println("*");
    } else {
        System.out.println();
    }
    System.out.println("    " + artist);
    System.out.println("    tracks: " + numberOfTracks);
    System.out.println("    " + comment);
}
/**
 * The Video class represents a video object. Information about the
 * video is stored and can be retrieved.
 *
 * @author Michael Kolling and David J. Barnes
 * @version 2002-05-02
 */

public class Video {
    private String title;
    private String director;
    private int playingTime;
    private boolean gotIt;
    private String comment;

    /**
     * Constructor for objects of class Video
     */
    public Video(String theTitle, String theDirector, int time) {
        title = theTitle;
        director = theDirector;
        playingTime = time;
        gotIt = false;
        comment = "<no comment>";
    }

    // methods go here
}
public void setComment(String comment) {
    this.comment = comment;
}

public String getComment() {
    return comment;
}

public void setOwn(boolean ownIt) {
    gotIt = ownIt;
}

public boolean getOwn() {
    return gotIt;
}

public void print() {
    System.out.print("video: "+ title + ", "+ playingTime + " mins");
    if(gotIt) {
        System.out.println("*");
    } else {
        System.out.println();
    }
    System.out.println("    " + director);
    System.out.println("    " + comment);
}
An Improved Version

• So now the question arises how DoME can be improved
• It is clear that CDs and videos share many values
• Specifically, what values can be abstracted to a parent class?
• What values remain specific to CDs?
• What values remain specific to videos?
• How else can we improve the structure of the DoME application?
import java.util.ArrayList;
import java.util.Iterator;

public class Database
{
    private ArrayList items;

    public Database()
    {
        items = new ArrayList();
    }

    public void addItem(Item theItem)
    {
        items.add(theItem);
    }

    public void list()
    {
        for(Iterator iter = items.iterator(); iter.hasNext(); )
        {
            Item item = (Item)iter.next();
            item.print();
        }
    }
}
/**
 * The Item class represents a multi-media item.
 * Information about the item is stored and can be retrieved.
 * This class serves as a superclass for more specific items.
 *
 * @author Michael Kolling and David J. Barnes
 * @version 2002-05-04
 */
public class Item
{
    private String title;
    private int playingTime;
    private boolean gotIt;
    private String comment;

    /**
     * Initialise the fields of the item.
     */
    public Item(String theTitle, int time)
    {
        title = theTitle;
        playingTime = time;
        gotIt = false;
        comment = "";
    }
    // methods go here
}
/**
 * Enter a comment for this item.
 */
public void setComment(String comment)
{
    this.comment = comment;
}

/**
 * Return the comment for this item.
 */
public String getComment()
{
    return comment;
}

/**
 * Set the flag indicating whether we own this item.
 */
public void setOwn(boolean ownIt)
{
    gotIt = ownIt;
}
/**
 * Return information whether we own a copy of this item.
 */

public boolean getOwn()
{
    return gotIt;
}

/**
 * Print details about this item to the text terminal.
 */

public void print()
{
    System.out.print("title: " + title + " (" +
                  playingTime + " mins)");

    if(gotIt) {
        System.out.println("*");
    } else {
        System.out.println();
    }
    System.out.println("    " + comment);
}
/**
 * The CD class represents a CD object. Information about the CD is stored and can be retrieved.
 * 
 * @author Michael Kolling and David J. Barnes
 * @version 2002-05-04
 */

public class CD extends Item {
    private String artist;
    private int numberOfTracks;

    public CD(String theTitle, String theArtist, int tracks, int time) {
        super(theTitle, time);
        artist = theArtist;
        numberOfTracks = tracks;
    }

    public String getArtist() {
        return artist;
    }

    public int getNumberOfTracks() {
        return numberOfOfTracks;
    }

    CD
/**
 * The Video class represents a video object. Information about the
 * video is stored and can be retrieved.
 *
 * @author Michael Kolling and David J. Barnes
 * @version 2002-05-04
 */

public class Video extends Item
{
    private String director;

    /**
     * Constructor for objects of class Video
     */
    public Video(String theTitle, String theDirector, int time)
    {
        super(theTitle, time);
        director = theDirector;
    }

    /**
     * Return the director for this video.
     */
    public String getDirector()
    {
        return director;
    }
}
Some Details on Subtyping

• Consider Car a subclass of Vehicle

```java
Vehicle v; Car c;
c = new Car();
v = c;  // OK because all cars are vehicles
c = (Car) v;  // this works because v is a Car subtype
```

• If you are not sure of the subtype, you can always check it directly

```java
if(v instanceof Car)
    c = (Car) v;
else if(v instanceof Bicycle)
    b = (Bicycle) v; // assuming b is of type Bicycle
//etc
```

• “Downcasting” should always be done with an explicit test of the subtype
Wrapper Classes

• We have already seen two uses of wrapper classes
  – Character is the wrapper class for char
  – Integer is the wrapper class for int
• All of the primitive types in Java have corresponding wrapper classes that are objects
• Remember, Collections can only store Objects but arrays can store both primitive types and Objects
• Java 1.5 introduces auto-wrapping which makes converting primitives to/from Objects easier and, in particular, it is easier to store primitive types in a collection since they are “auto-wrapped”