CM 10227: Lecture 7

Dr. Rachid Hourizi and Dr. Michael Wright

December 8, 2016

Resources

- More help with this course
  - Moodle
  - E-mail - programming1@lists.bath.ac.uk
- Online C and Java IDE
  - `https://www.codechef.com/ide`
  - Remember to select Java from the drop down menu.
- Books
  - Java by Dissection (Free pdf online)
  - The Java Tutorial (https://docs.oracle.com/javase/tutorial/)

- The places that you can get additional support if you are finding the pace of the course a little fast now include
  - The A labs
  - The B (catch up) lab
  - The Drop in Sessions
- please note that we have moved a couple of the labs
- please check the details on Moodle and let us know if you now cannot get to a lab that you wold otherwise have attended.

- If you struggling with the exercises, pace of the course and/or coding in general
- Please come and see Rachid or Michael

- If, on the other hand, you are finding the pace a little slow
- You can still sign up for the Advanced Programming Labs

Last time...

- First Classes and Objects

This week

- (Almost) Reusing Code
  - Inheritance
  - Polymorphism

A Brief Recap

- Java Classes (Templates)
  - Fields
  - Constructors
  - Accessors and Mutators (Sometimes called Getters and Setters)

```
public class Example{

}
```

```
public class Example{
        private int exampleVariable;
}
```

```
public class Example{
        private int exampleVariable;

        public Example(){
                exampleVariable = 1;
        }
}
```

```java
public class Example{
        private int exampleVariable;

        public Example(){
                exampleVariable = 1;
        }

        public Example(int value){
                exampleVariable = value;
        }
}
```

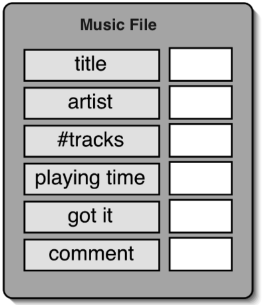
```
public class Example{
        private int exampleVariable;

        public Example(){
                exampleVariable = 1;
        }

        public Example(int value){
                exampleVariable = value;
        }

        public void setExampleVariable(int value){
                exampleVariable = value;
        }
}
```

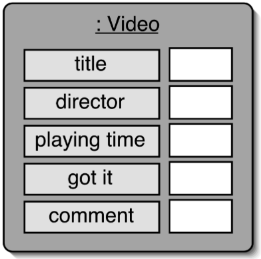
```
public class Example{
        private int exampleVariable;

        public Example(){
                exampleVariable = 1;
        }

        public Example(int value){
                exampleVariable = value;
        }

        public void setExampleVariable(int value){
                exampleVariable = value;
        }

        public int getExampleVariable(){
                return exampleVariable;
        }
}
```

Inheritance

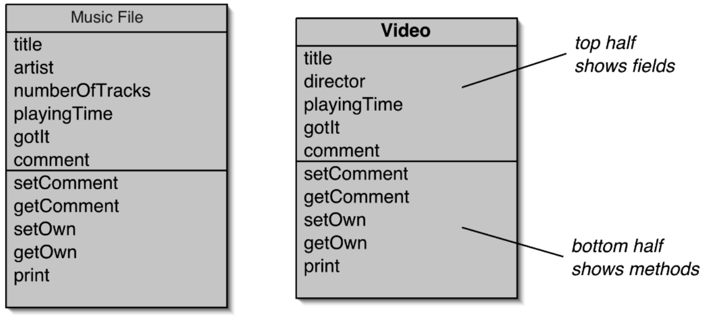
- **Database of Multimedia Entertainment**
- Stores details about Music Files and videos
  - Music File: title, artist, # tracks, playing time, got-it, comment
  - Video: title, director, playing time, got-it, comment
- Allows (later) to search for information or print lists

DOME Object Diagram

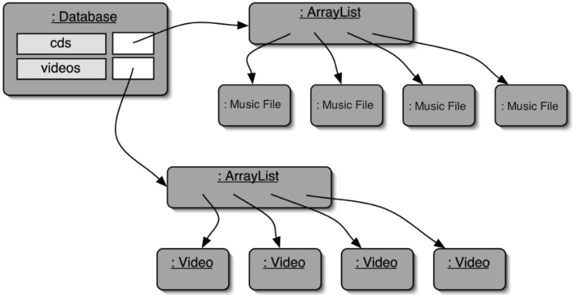




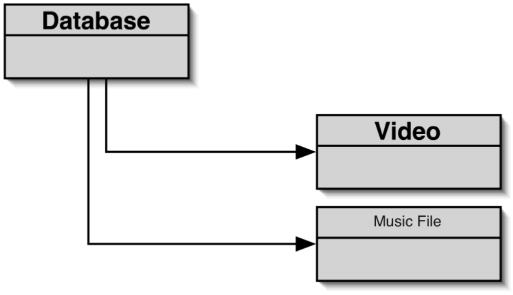
DOME Class Diagram



DOME Object Diagram continued...



DOME Class Diagram continued...



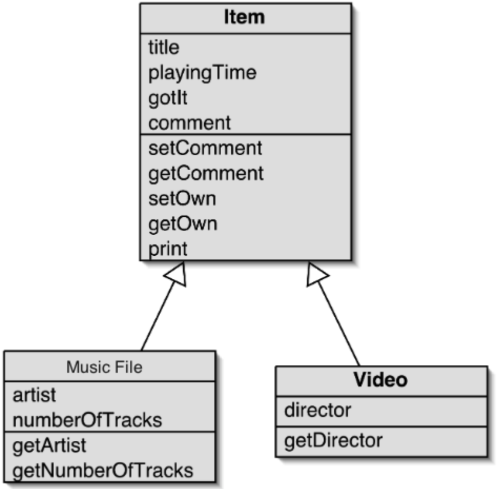
```
public class MusicFile {
  private String title ;
  private String artist ;
  private String comment;

  public MusicFile( String theTitle , String theArtist
      ) {
  title = theTitle;
  artist = theArtist;
  comment = "";
  }

  public void setComment (String newComment)  {...}

  public String getComment()  {...}

  public void print ()  {...}

}
```

```java
public class Video{
  private String title ;
  private String director ;
  private String comment;

  public Video( String theTitle, String theDirect ){
    title = theTitle;
    director=theDirect;
    comment="";
  }

  public void setComment(String newComment) { ... }

  public String getComment() { ... }

  public void print()  { ... }

}
```

```
public class Database{
  private ArrayList MusicFiles;
  private ArrayList videos;

  public void list(){

    for(i=0, i<MusicFiles.size(); i++){
      MusicFile.get(i).print();
      System.out.println();
    }

    for(i=0, i<videos.size(); i++){
      videos.get(i).print();
      System.out.println();
    }
  }
}
```

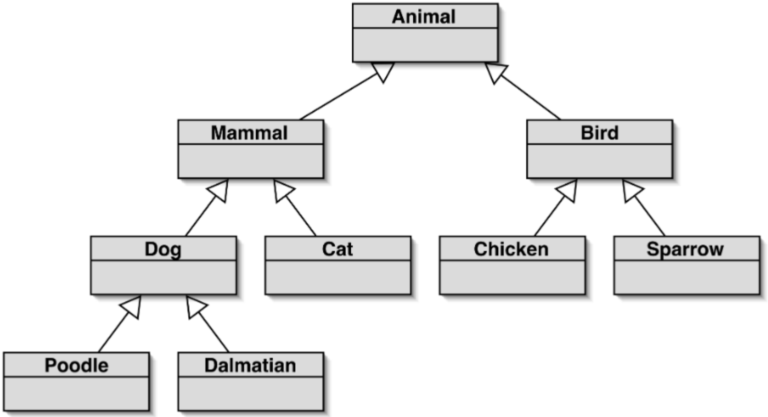
- Critique of DOME
- Code duplication
  - MusicFile and Video classes very similar (large part are identical)
  - makes maintenance difficult/more work
  - introduces danger of bugs through incorrect maintenance
- Code duplication also in Database class

DOME Object Diagram Using Inheritance



- Using Inheritance
    - define one superclass : Item
    - define subclasses for Video and MusicFile
    - the superclass defines common attributes
    - the subclasses inherit the superclass attributes
    - the subclasses add own attributes

Object Diagram Using Inheritance



```java
public class Item{
  private String title ;
  private int playingTime ;
  private boolean gotIt ;
  private String comment;

  //constructors and methods omitted...
}
```

```java
public class MusicFile extends Item{

  private String artist;
  private int numberOfTracks;

  //constructors and methods omitted

}
```

```java
public class Video extends Item{

  private String director;

  //constructors and methods omitted

}
```

```
public class Item{

  private String Title;
  private int playingTime;
  private boolean gotIt;
  private String comment;

  public Item(Sring theTitle, int time){
    title = theTitle;
    playingTime = time;
    gotIt = false;
    comment = "";
  }

  //methods omitted

}
```

```java
public class MusicFile extends Item{
  private String artist;
  private int numberOfTracks;

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

  //methods omitted

}
```

- 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.

```java
public class Database {
  private ArrayList<Item> items ;

  // Construct an empty Database
  public Database( ) {
    items = new ArrayList<Item>() ;
  }

  // Add an item to the database
  public void addItem ( Item theItem ) {
    items.add(theItem);
  }
  ...
}
```

http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html

```
/**
* Print a list of all currently stored MusicFiles and
* videos to the text terminal .
**/

public void list () {
  for(i=0; i<items.size; i++){
    Item item = items.get(i);
    item.print();
    System.out.println() ;
  }
}
```

- Subtyping
- First, we had:
  - public void addMusicFile(MusicFile theMusicFile)
  - public void addVideo(Video theVideo)
- Now, we have:
  - public void addItem(Item theItem)
  - We call this method with:

```
Video myVideo = new Video(...);
database.addItem(myVideo);
```

Subclasses and subtyping

- Classes define types.
- Subclasses define subtypes.
- Objects of subclasses can be used where objects of supertypes are required.
- This is called substitution.

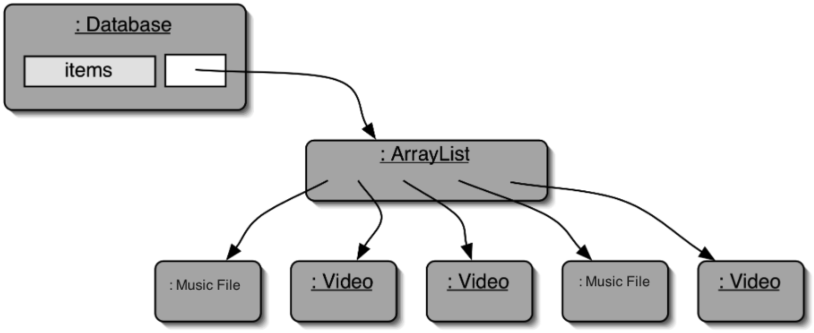
- Subclass objects may be assigned to superclass variables
- e.g. Car extends Vehicle and Bicycle extends Vehicle

```
Vehicle v1 = new Vehicle();
Vehicle v2 = new Car();
Vehicle v3 = new Bicycle();
```

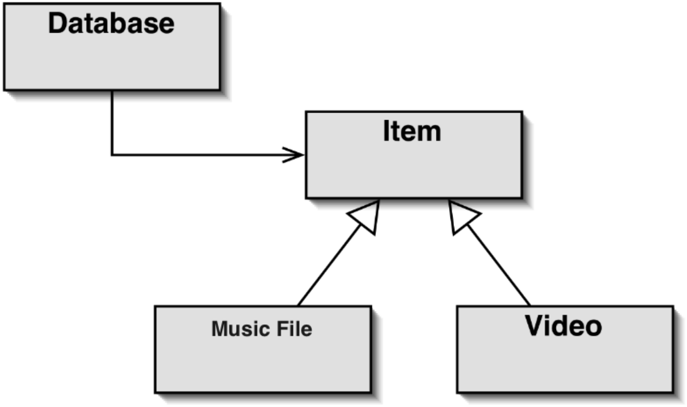
- Subclass objects may be passed to superclass parameters

```
public class Database{
  public void addItem (Item theItem){
    ....
  }
}

//code in another method
Video video = new Video(...);
MusicFile MusicFile = new MusicFile(...);

database.addItem (video);
database.addItem (MusicFile);
```

Object Diagram Illustrating Inheritance



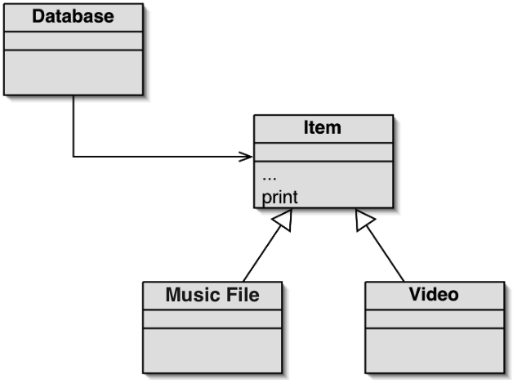
Class Diagram Illustrating Inheritance



Review

- Inheritance allows the definition of classes as extensions of other classes.
- Inheritance
  - avoids code duplication
  - allows code reuse
  - simplifies the code
  - simplifies maintenance and extending
- Variables can hold subtype objects.
- Subtypes can be used wherever supertype objects are expected (substitution).

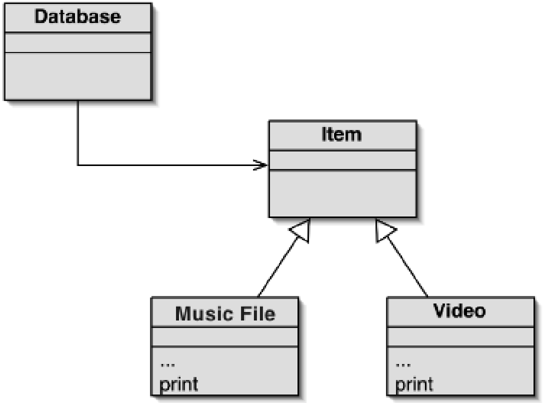
- Polymorphic variables
- Object variables in Java are polymorphic.
  - They can hold objects of more than one type.
- They can hold objects of the declared type, or of subtypes of the declared type.


Database
Item
...
print
Music File
Video

- The print method in Item only prints the common fields.
- Inheritance is a one-way street:

- A subclass inherits the superclass fields.
- The superclass knows nothing about its subclasss fields.

- Attempting to Solve the Problem.
- Place print where it has access to the information it needs.



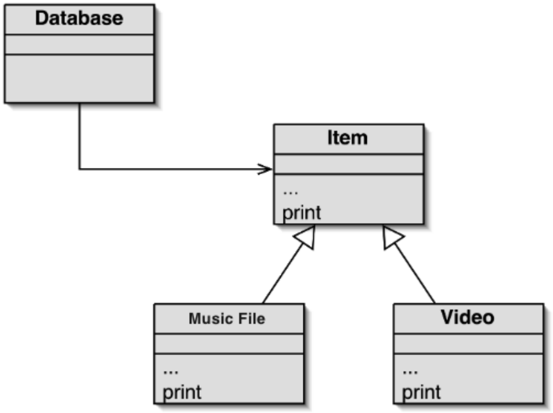
- Each subclass has its own version.
- But Items fields are private.
- Database cannot find a print method in Item.

- To solve our problem we need to introduce...
- some new terminology:
  - static type
  - dynamic type
  - method dispatch/lookup

- Static Type and Dynamic Type
- The declared type of a variable is its static type.
- The type of the object a variable refers to is its dynamic type.
- The compilers job is to check for static-type violations.

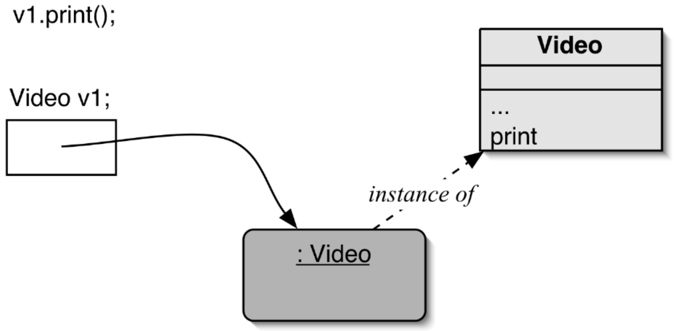
```
class Alpha{}
class Beta extends Alpha{}
class Fruit extends Beta{}

Fruit f = new Fruit(); //static=Fruit, dynamic=Fruit
Alpha a = f; //static=Alpha, dynamic=Fruit

Fruit f = a //static type violation
```

- Returning to our problem...
- The Solution: Overriding
  - print method in both super- and subclasses
  - Satisfies both static and dynamic type checking

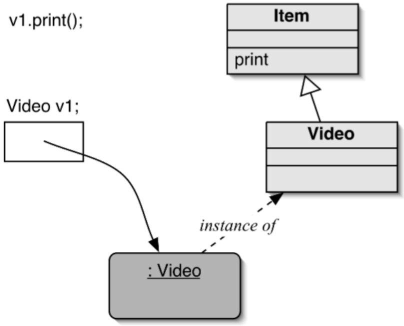

Database
Item
...
print
Music File
...
print
Video
...
print

- Superclass and subclass define methods with the same signature.
- Each has access to the fields of its class.
- Superclass satisfies static type check.
- Subclass method is called at runtime it overrides the superclass version.
- What becomes of the superclass version?

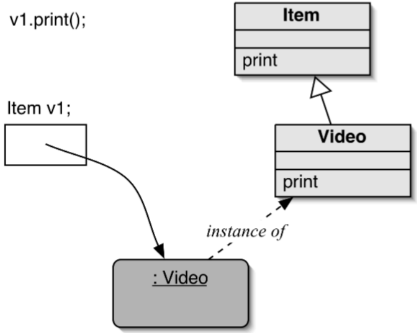
- Method Lookup 1
  - No inheritance or polymorphism.
  - The obvious method is selected.



- Method Lookup 2
  - Inheritance but no overriding
  - The inheritance hierarchy is ascended, searching for a match.



- Method Lookup 3
  - Polymorphism and overriding.
  - The first version found is used.



Method Lookup Summary

- The variable is accessed.
- The object stored in the variable is found.
- The class of the object is found.
- The class is searched for a method match.
- If no match is found, the superclass is searched.
- This is repeated until a match is found, or the class hierarchy is exhausted.
- Overriding methods take precedence.

- Super call in methods
- Overridden methods are hidden ...
- ... but we often still want to be able to call them.
- An overridden method can be called from the method that overrides it
    - super.method(...)
    - Compare with the use of super in constructors.

```java
public class MusicFile{

  ...

  public void print (){
    super.print();
    System.out.println (""+artist);
    System.out.println("tracks:" + numberofTracks);
  }
}
```

- We have been discussing polymorphic method dispatch.
- A polymorphic variable can store objects of varying types.
- Method calls are polymorphic.
    - The actual method called depends on the dynamic object type.

- Methods in Object are inherited by all classes.
- Any of these may be overridden.
- The toString method is commonly overridden:
- public String toString()
  - Returns a string representation of the object.

```
public class Item{

 ...

  public String toString (){
    String line1=title + "␣:␣" + playingTime + "␣mins"
       );
    if(gotIt) {
      return line1 + "\n" + comment + "\n");
    }
    else {
      return line1 + "\n" + comment + "␣need␣to␣buy" +
            "\n");
    }
  }
}
```

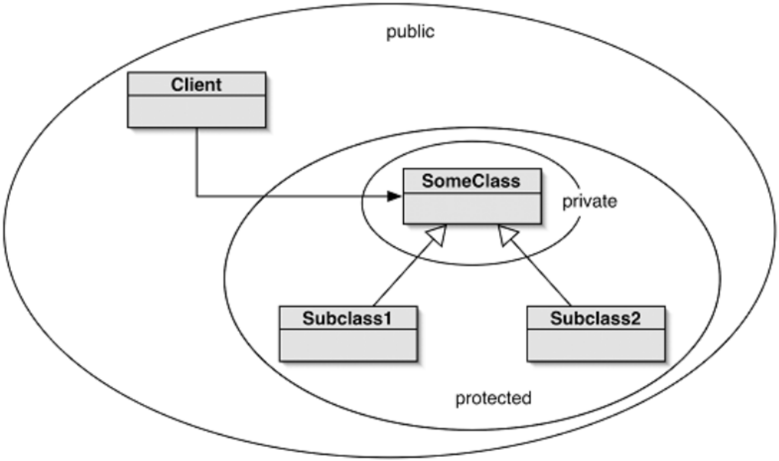
- Explicit print methods can often be omitted from a class:

```
System.out.println(item.toString()) ;
```

- Calls to println with just an object automatically result in
- toString being called:

```
System.out.println(item);
```

- Private access in the superclass may be too restrictive for a subclass.
- The closer inheritance relationship is supported by protected access.
- Protected access is more restricted than public access.
- We still recommend keeping fields private.
- Define protected accessors and mutators.


public
Client
SomeClass
private
Subclass1
Subclass2
protected

- Review

- The declared type of a variable is its static type.
- Compilers check static types.
- The type of an object is its dynamic type.
- Dynamic types are used at runtime.
- Methods may be overridden in a subclass.
- Method lookup starts with the dynamic type.
- Protected access supports inheritance.