Object Orientation with Design Patterns

Lecture 1 : Objects & Classes revision

Objects

- Object-oriented programming is modelled on how, in the real world, **objects are often made up of many kinds of smaller objects.**

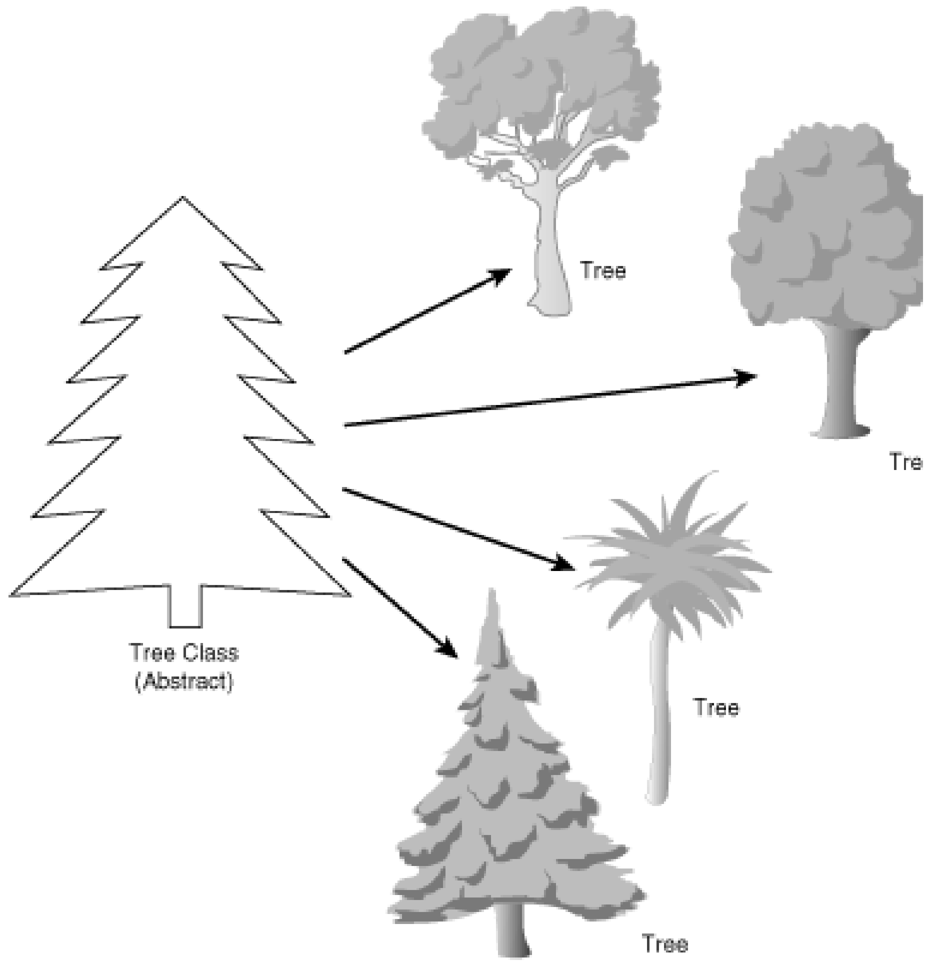
- This capability of combining objects, however, is only **one very general aspect** of object-oriented programming. Object-oriented programming provides several other concepts and features to make creating and using objects easier and more flexible, and the most important of these features is classes.

- When you write a program in an object-oriented language, you don't define actual objects. You **define classes of objects**, where a class is a **template** for multiple objects with similar features.

Objects

- Classes embody all the features of a particular set of objects.
- For example, you might have a Tree class that describes the features of all trees (has leaves and roots, grows, creates chlorophyll). The Tree class serves as an abstract model for the concept of a tree-to reach out and grab, or interact with, or cut down a tree you have to have a concrete instance of that tree. Of course, once you have a tree class, you can create lots of different instances of that tree, and each different tree instance can have different features (short, tall, bushy, drops leaves in autumn), while still behaving like and being immediately recognizable as a tree

Objects



Objects

- Definition of a class
  - A class is a generic template for a set of objects with similar features.

- An **instance** of a class is another word for an **actual object**.

- If **class** is the general **(generic) representation** of an object, an **instance** is its **concrete representation.**

- So what, precisely, is the difference between an instance and an object? **Nothing, really**. Object is the more general term, but both instances and objects are the **concrete representation** of a class. In fact, the terms instance and object are often used interchangeably in OOP lingo. An instance of a tree and a tree object are both the same thing.

Objects

- Definition of an Instance
  - **An instance is the specific concrete representation of a class. Instances and objects are the same thing.**

- Another way of distinguishing classes from objects is as follows:
  - **A *class* exists at design time**, i.e. when we are writing OO code to solve problems.
  - **An *object* exists at runtime** when the program that we have just coded is running.

- This distinction will become much clearer later on. For now lets look at writing a very simple class in Java.

Objects

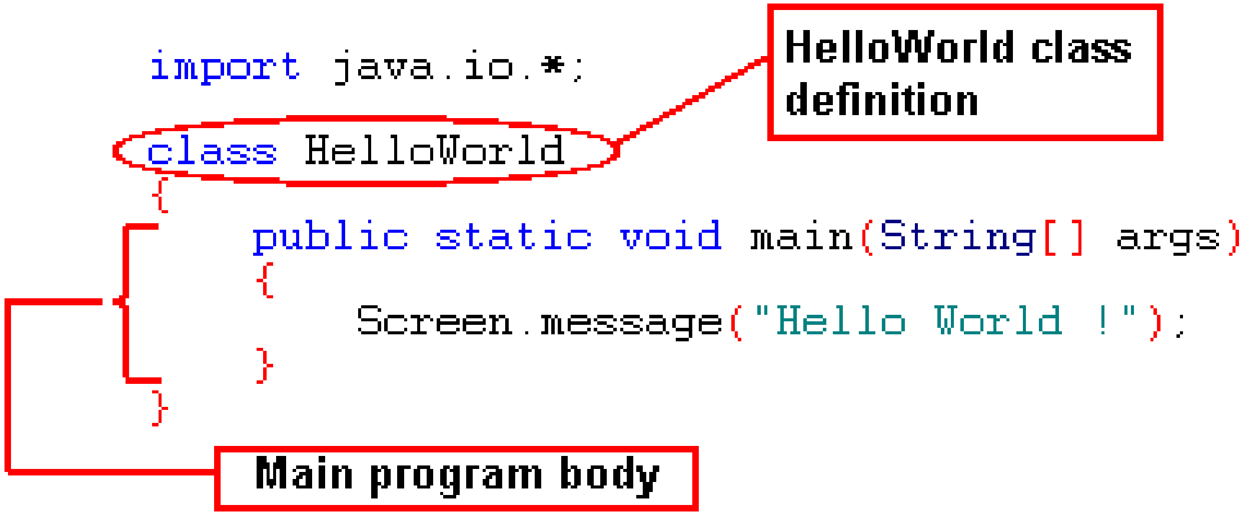
- Lets say for example you wanted to write a program to store information on your CD collection. You may want to store certain information on each CD such as:
  - Title
  - Artist
  - Number of tracks
  - Favourite track

- This can be easily achieved in Java by writing a class called **CompactDisk** which will represent the abstract concept of a CD.

- We will then be able to create as much CompactDisc objects as we like in order to model our CD collection. All of these objects will be an instance of our CompactDisc class.

Writing classes in Java

- We have actually written some classes in Java already. It is impossible to write even a simple program in Java without having to write a class.
- All of the simple programs that we wrote in lecture 1 were classes. In Java even your program is a class, lets look at an example.



```
import java.io.*;
class HelloWorld
{
    public static void main(String[] args)
    {
        Screen.message("Hello World !");
    }
}
```

Writing classes in Java

- In the HelloWorld program on the previous slide we can see the line of code which reads:

  class HelloWorld

- Here we are actually creating a new class which will represent our Java program.

- As was stated in the first lecture the **Java programming language is truly object oriented**, there’s **no getting away from classes** and there’s certainly no getting away from objects.

The class construct

- It is essential that we understand and can use the class construct in Java. As we have seen in previous examples even our programs are defined by a classes.

```
// Basic Class Construct
class [class name]
{
    // Attributes / identity Section


    // Methods / behavior Section
}
```

- There are a number of important aspects of object oriented programming which we must address before we can write any useful classes such as the CompactDisc class.
    - Class Attributes
    - Class Methods/operations
    - Data Hiding & Encapsulation

Attributes

- When we try to model objects in the real world we usually pick **certain aspects** about them which **uniquely make** them what they are.

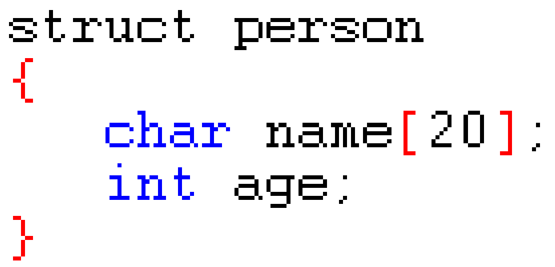
  For example if we were to describe a **car** we may choose the following aspects:
    - **Make, Model, Colour, Engine Capacity, Year, Reg**

  If we were to model a **table** we might choose the following aspects:
    - **Width, Length, Height, No. Legs**

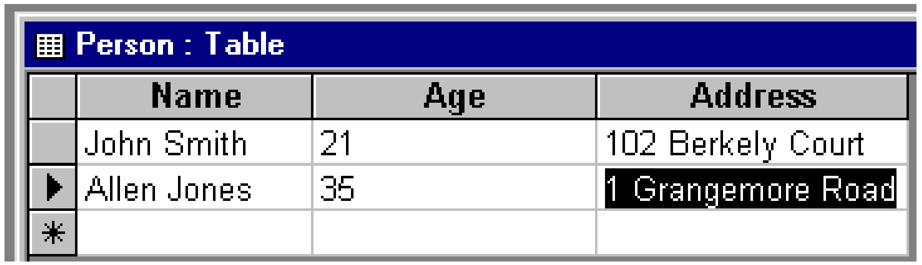
- In the context of object oriented programming all of the above examples would be attributes of a car or table object.

Attributes

- So an **object is defined by its attributes**
- You can think of an objects attributes as being like fields in a database table or fields in a structure that is part of a 'C' program.



```
struct person
{
    char name[20];
    int age;
}
```



Person : Table

	Name	Age	Address
	John Smith	21	102 Berkely Court
▶	Allen Jones	35	1 Grangemore Road
✳			

Methods/operations

- Not only does an object have a **definition** which is given by its **attributes**, it can also have a **behaviour.**

- For example a **dog** object may have the following **behaviours:**
  - **Walk, sleep, run, eat, drink, sh*t…. And so on**

- A **car** object may have the following behaviours:
  - **Start engine, stop engine, engage clutch, change gears…. And so on**

- We can add behaviour to our objects by including methods in their class definition. **Methods** or operations are just another **name for functions** which are defined inside the class.

Data hiding & Encapsulation

- Encapsulation and data hiding bind data and procedures (operations) together and limit the scope and visibility of the procedures and functions that can manipulate the data to a highly localized region of code in the software system(the class).

- **The data and related procedures become inseparable and a new entity called an object is defined.**

- ***Encapsulation*** is the **binding of the underlying data(attributes)** with **an associated set of procedures** and **functions(behavior)** that **can manipulate the data**. (Attributes of object can be manipulated ONLY by methods of the object- we close off our classes and protect attributes from outside interference)

- ***Data hiding*** is the **inaccessibility** of the **internal structure of the underlying data**.(more on this later)

Data hiding & Encapsulation

- Objects are defined in terms of classes. A class description defines the characteristics of all objects declared to be of a given class. The **class construct also provides the basic unit of reusability in Java.**

- It binds the underlying data type with a set of methods (functions/interface) that allow the underlying data type to be manipulated by passing messages to the object (calling its functions).

- Data hiding is achieved by using ***access modifiers*** to **limit the visibility of an objects attributes and methods to the outside world**. There are two access modifiers that we should be aware of at this point, ***private*** and ***public***.

Creating a class

- The best way to get comfortable with all of this new knowledge is to write a class of our own. So lets have a go at writing the CompactDisc class.

- Let's start with a basic class definition. Open the text editor you've been using to create Java source code and enter the following (remember, upper- and lowercase matters):

```
class CompactDisc
{

}
```

- Congratulations! You've now created a class. Of course, it doesn't do very much at the moment, but that's a Java class at its very simplest.

Creating a class

- Now lets add some attributes to the class definition

```
class CompactDisc
{
    // Attributes : Object identity
    private String title;
    private String artist;
    private int num_tracks;
    private String favorite_track;

}
```

- Now that our class has some attributes we can begin to define some of its behaviors. We will need to be able to set values for all of its attributes and we will also need to be able to query it for an attribute value. For example we might want to give a CD object the title “White Ladder” and the artist “David Gray”. We may also want to be able to ask an object, what’s your title? What’s your artist and so on.

Creating a class

- Remember in order to add behavior all we have to do is add some functions to the class definition. So for example we could add the following method to set a CD's title.

```
class CompactDisc
{
    // Attributes : Object identity
    private String title;
    private String artist;
    private int num_tracks;
    private String favorite_track;

    // Methods : Object behavior
    public void setTitle(String t)
    {
        title = t;
    }
}
```

Creating a class

- The best way to see how all of this works is to write a small test program which will run our CompactDisc class through its paces.
- Open the text editor you've been using to create Java source code and enter the following. This file must be saved in the same directory as the CompactDisc.java file.

Creating a class

```
class CDTest
{
    public static void main(String[] args)
    {
        // Create 2 object variables of
        // type CompactDisc
        CompactDisc cd1;
        CompactDisc cd2;
                                          Class
        // Allocate some memory for
        // each new object
        cd1 = new CompactDisc();
        cd2 = new CompactDisc();

        // Set each of the CD's titles
        // using the setTitle method
        cd1.setTitle("Kid A");
        cd2.setTitle("The Bends");
    }                                     Object
}
```

Creating a class

- If you compile and run this test program it does absolutely nothing, why?
- All we have done is created two CompactDisc objects and set their titles.
- How do we know whether each CD object cd1, cd2, has received the given titles?
- By adding a second method to query the CD object for its title we will be able to assure ourselves that each objects attribute has been set successfully.

Creating a class

```
class CompactDisc
{
    // Attributes : Object identity
    private String title;
    private String artist;
    private int num_tracks;
    private String favorite_track;

    // Methods : Object behavior
    public void setTitle(String t)
    {
        title = t;
    }
    public String getTitle()
    {
        return title;
    }
}
```

Creating a class

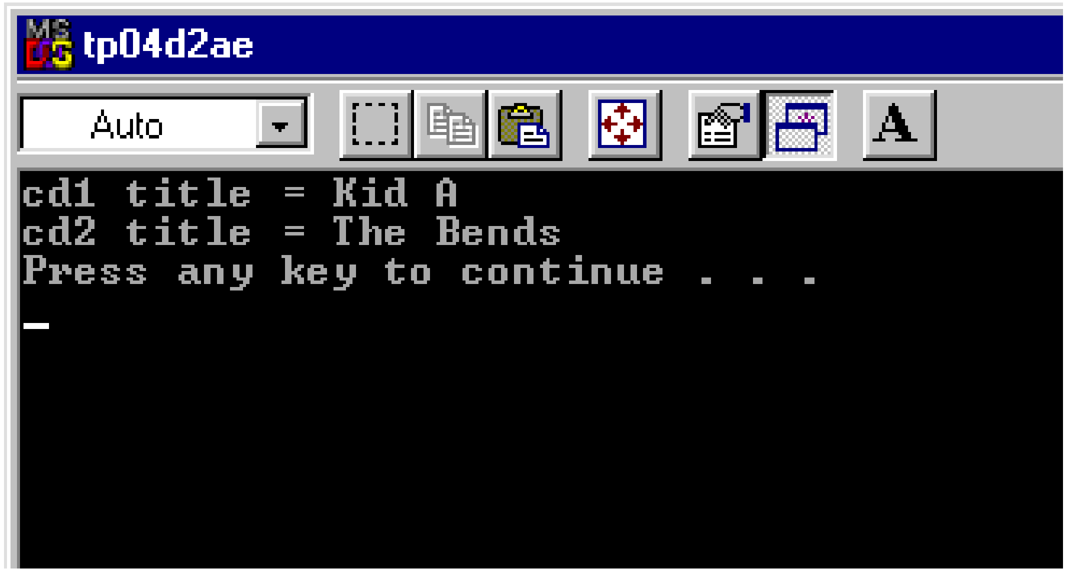
- The getTitle method returns the current value stored by the title attribute.
- Question : will this value be the same for each object ?
- Lets find out by modifying our test program. We can answer the question above by outputting the title values for cd1 and cd2 using the getTitle method.

Creating a class

```
class CDTest
{
    public static void main(String[] args)
    {
        // Create 2 object variables of
        // type CompactDisc
        CompactDisc cd1;
        CompactDisc cd2;

        // Allocate some memory for
        // each new object
        cd1 = new CompactDisc();
        cd2 = new CompactDisc();

        // Set each of the CD's titles
        // using the setTitle method
        cd1.setTitle("Kid A");
        cd2.setTitle("The Bends");

        // Output each objects title value
        Screen.message("cd1 title = " + cd1.getTitle());
        Screen.newline();
        Screen.message("cd2 title = " + cd2.getTitle());
        Screen.newline();
    }
}
```

Creating a class

- When we compile and run the test program now we get the following output



```
cd1 title = Kid A
cd2 title = The Bends
Press any key to continue . . .
_
```

- So the answer to the question is no, each object will be able to store its own set of attribute values. This means that even though we have created the two objects using same class they will both have their own personal copy of data.

Creating a class

- Now that we are happy that our CompactDisc class is working we can add set and get methods for each attribute. All of these methods will be based on the setTitle and getTitle methods.
  - Full compliment of Set methods

```
public void setTitle(String t)
{
    title = t;
}
public void setArtist(String a)
{
    artist = a;
}
public void setNumTracks(int n)
{
    num_tracks = n;
}
public void setFavoriteTrack(String t)
{
    favorite_track = t;
}
```

Creating a class

– Full compliment of Get methods

```
public String getTitle()
{
    return title;
}
public String getArtist()
{
    return artist;
}
public int getNumTracks()
{
    return num_tracks;
}
public String getFavoriteTrack()
{
    return favorite_track;
}
```

Private & Public access modifiers

- As we mentioned earlier, Data hiding is one of the most important features of object oriented programming. **Data hiding is achieved through the use of the Java keyword *private***.

- If you examine the CompactDisc class you will see that all the **attribute declarations are preceded by the keyword private**. This tells the java compiler that the attribute variables can only be referenced within the scope of the CompactDisc class.

- So the attribute variables can only be accessed directly within the CompactDisc class. If we need to access the attribute variables from outside of the class, i.e. in the test program we must use one of the classes get methods.