Object Oriented Analysis and Design (OOAD)

Objects & Classes

COMP H2031

Lecturer: Frances Murphy

Objectives

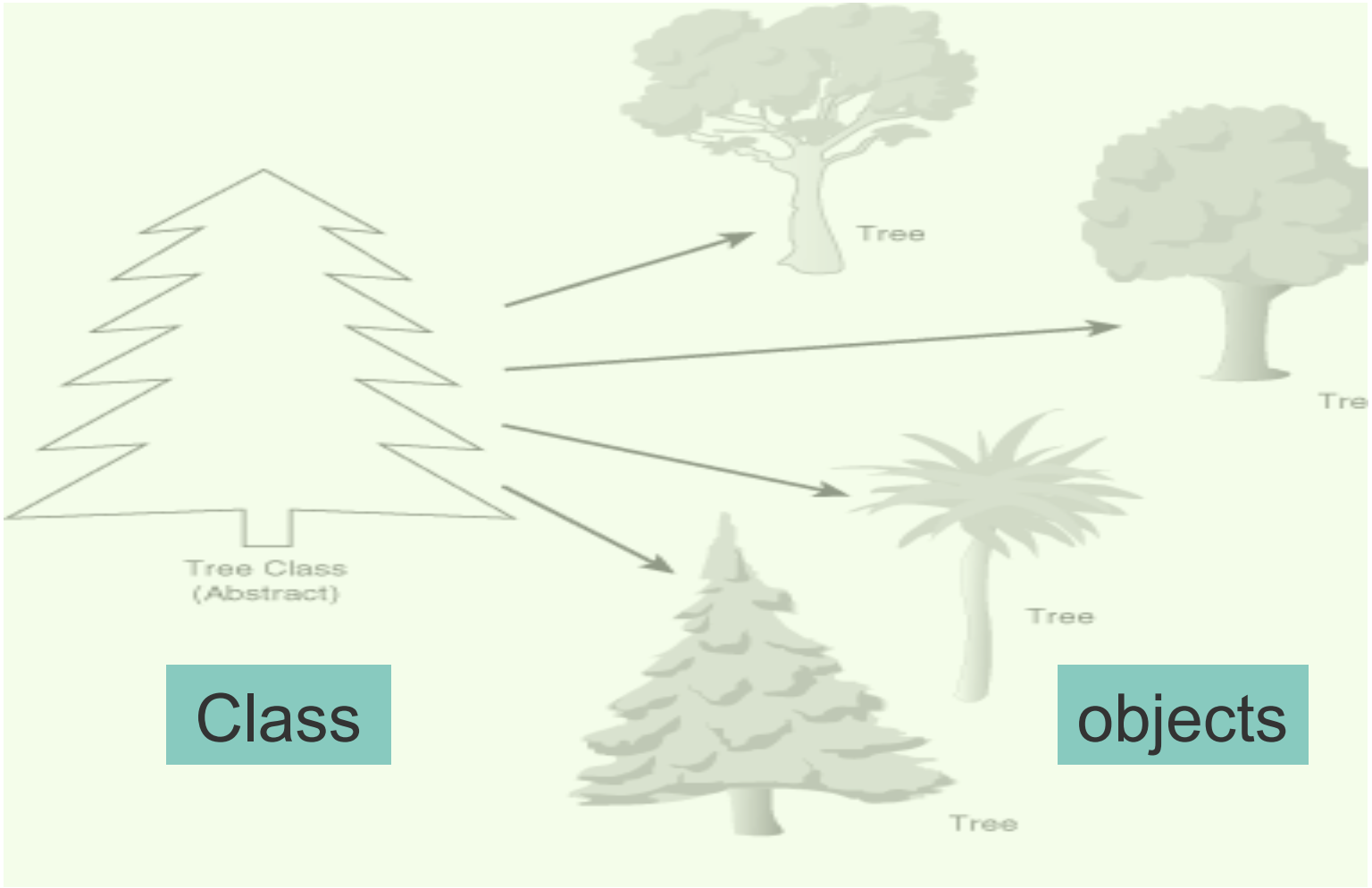
At the end of this lecture, you should be able to:

- Distinguish between a Class, Object and Instance of a Class
- Draw an UML Diagram of a class
- Write a Class and Class Tester Program
- Write a program using Default and User-Defined Constructors
- Define various OO terms

Recap --- Class

- Classes embody all the features of a particular set of objects.
- Tree class describes the features of <u>all</u> trees
- Once you have a tree class, you can create lots of different instances of that tree
- Each different tree instance can have different features (short, tall, bushy, evergreen, deciduous, apple tree, cherry tree)

Class & Objects



Definition – Class & Object

- Definition of a class
  - A class is a generic template for a set of objects with similar features.
- Definition of an Instance
  - An instance is the specific concrete representation of a class.
- An object is an instance of a class.

Definition – Class & Object

- Another way of distinguishing classes from objects:
  - 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.

Making an Object

one class

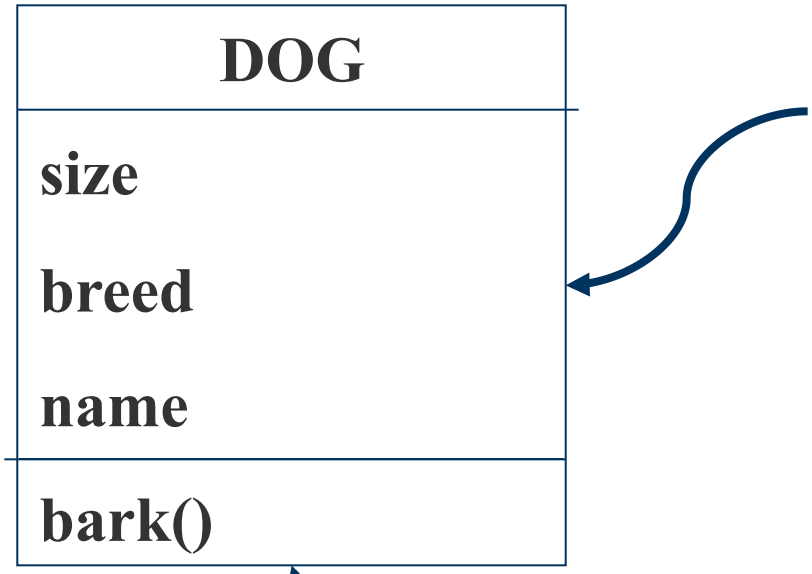
DOG
size breed name
bark()

instance variables – things an object knows about itself

operations / methods – things an object can do

Many “dog” objects

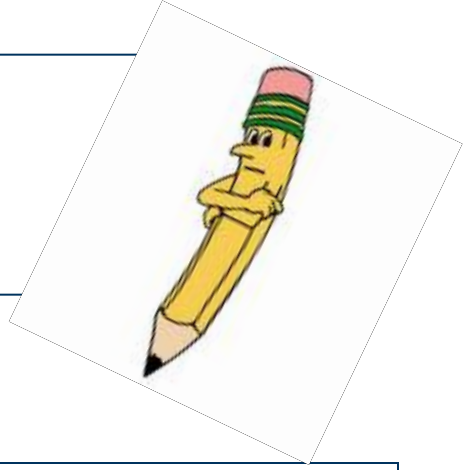
Making an Object



instance variables: objects have state

methods: objects have behaviour

Sharpen your Pencil



- Fill in what a television object might need to know and do.

Television

instance variables

methods

Making an Object

You need to:

1. Write your Class
2. Write a Tester Class
3. In your Tester class, make an object and access the object's variables & methods

1. Write your class

```
class Dog {
int size;
String breed;
String name;

void bark()  {
System.out.println("Ruff!Ruff!");
        }
}
```

instance variables

method

DOG
size breed name
bark()

2. Write your tester class

```
class DogTester {
public static void main (String[] args) {
// dog test code goes here

// make a Dog object
// set the size of the Dog
// call the bark() method
  }
}
```

3. Use your Tester class

```
class DogTester {
public static void main (String[] args) {
Dog d = new Dog();
d.size = 40;
d.breed = “poodle”;
d.bark(); }
}
```

make a Dog object

use the dot operator(.)
to set the size of the Dog

Call its bark() method

Recap - Data Hiding & Encapsulation

- Encapsulation and data hiding are the centre pieces of OO programming.
- ***Encapsulation*** – seals the data (and internal methods) safely inside the "capsule" of the class, where it can be accessed only by trusted users (i.e., by the methods of the class).
- (Recall also: Car & engine example)
- Why do that?
- To keep the internal details of a class hidden from a user – called ***Data Hiding***

How is Data Hiding achieved?

By using *access modifiers* which limit the visibility of an object's attributes and methods to the outside world.

We should be aware of two types of **access modifiers**, (at this point)

- ***private*** &
- ***public***

Getters and Setters

- Getters

Send back, as a return value, the value or state of an instance variable

- Setters

Take a parameter's value and use it to set the value of an instance variable

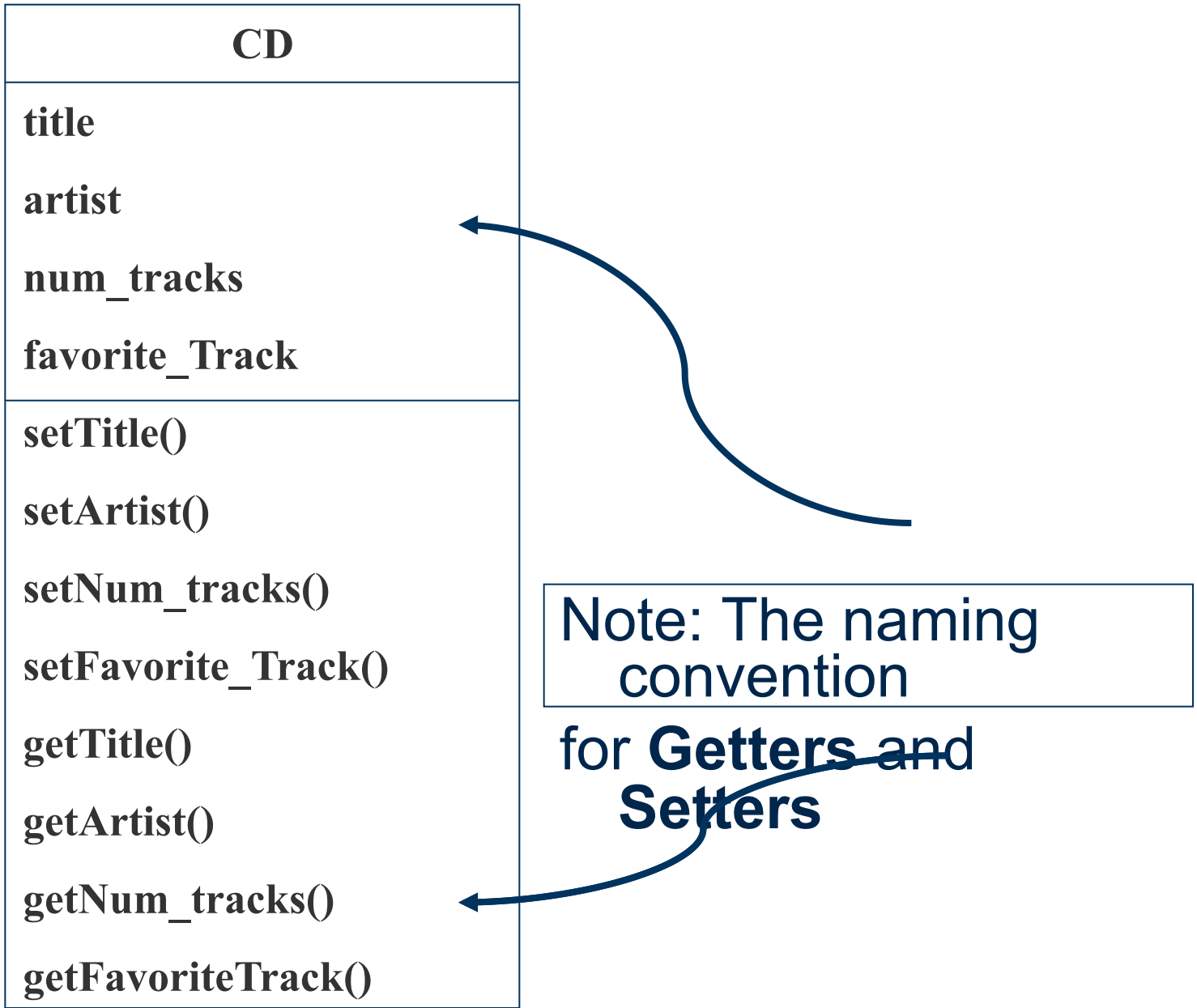
Encapsulation: Rule of Thumb

Mark your instance variables private.

Mark getters and setters public.

How is encapsulation achieved in a Java class?

Getters & Setters Naming Convention



Writing a Program – CD Class

- You want to write a program to store information on your CD collection and want to store certain information on each CD such as:
  - Title
  - Artist
  - Number of tracks
  - Favourite track

- Write a class called CompactDisc which will represent the abstract concept of a CD.
- Can then create as many CompactDisc objects as we like in order to model our CD collection.
- All of these objects will be an instance of our CompactDisc class.

Class Diagram

CD
title artist num_tracks favorite_Track
setTitle() setArtist() setNum_tracks() setFavorite_Track() getTitle() getArtist() getNum_tracks() getFavoriteTrack()

instance variables

Note: The naming convention

for Getters and Setters

operations / methods

Creating a Class

```
import java.io.*;
class CompactDisc
{
   // Attributes : Instance variables
  private String title;
  private String artist;
  private int num_tracks;
  private String favorite_track;
```

Creating a Class cont.

```
// 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 cont.

// Full set of Getter methods

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

Write a Tester Class

- Tester Class has the main() method
- Purpose is to create objects of the new type
- To use the dot operator (.) to access the methods and variables of the new objects

CD Class Test Program

```
class CDTest {
    public static void main(String[] args) {
      // Create 2 object variables of type CD
      CompactDisc cd1;
      CompactDisc cd2;
     // Allocate some memory for each new object
     cd1 = new CompactDisc();
     cd2 = new CompactDisc();
    // Set the title by accessing the attribute directly
    cd1.setTitle("Kid A");
    cd2.setTitle("The Bends");
   // Output each objects title value
    System.out.println("cd1 title = " + cd1.getTitle());
    System.out.println("cd2 title = " + cd2.getTitle());
   }
}
```

Constructors

1. Default Constructors
2. User Defined Constructors

Constructors

- Constructors are used to initialise an objects attributes, either by using:

1. a default constructor

   where the *initial values are hard coded*

or

2. a user defined constructor or overloaded constructor where *values can be passed to a class as parameters*

Points to Remember: Constructor

1. A constructor must have the same name as the class.

2. A constructor cannot have a return type declared, as it is not allowed to return any values (not even void).

The Role of Constructors

- Example 1:

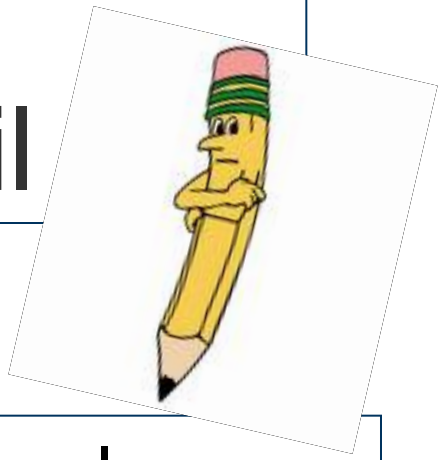
- Write a class that models an integer counter where the internal counter variable is initialised to zero when an object of that type was created.

1. Create the Counter Class

```java
class Counter {

    private int count;

    //default constructor
    Counter(){
        count = 0;
    }

    public void setCount(int val){
        count = val;
    }

    public int getCount(){
        return count;
    }

    public void increment(){
        count++;
    }

    public void decrement(){
        count--;
    }
}
```

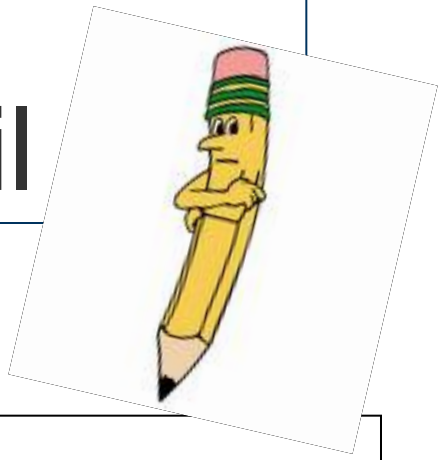
default constructor

Sharpen your Pencil



- Draw a Class Diagram of the Counter class

Sharpen your Pencil



- The Default Constructor _____ a parameter list (has / hasn't).

- If you don't put a constructor in your class, the compiler will put in a _____ constructor.

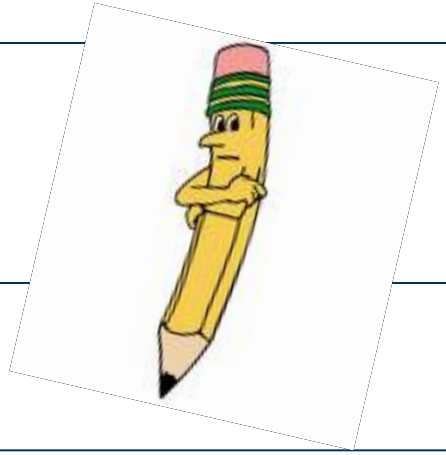
- You can put in a constructor to initialise the _____ of the object being constructed.

2. Test Program

```
class TestCounter1 {

    public static void main(String[] args){

        //declare two counter objects
        Counter c1 = new Counter();
        Counter c2 = new Counter();

        //test the value of the counter objects
        System.out.println("Values after construction:");
        System.out.println("Counter 1: "+c1.getCount());
        System.out.println("Counter 2: "+c2.getCount());
        System.out.println();

        //test the increment() and decrement()
        c1.increment();
        c2.decrement();

        //test the value of the counter objects
        System.out.println("Values after construction:");
        System.out.println("Counter 1: "+c1.getCount());
        System.out.println("Counter 2: "+c2.getCount());
    }
}
```

constructors get called

Output



```
Values after construction:
Counter 1:  ____
Counter 2: ____

Values after construction:
Counter 1:  ____
Counter 2: ____

Process completed
```

The Role of Constructors

- Example 2:

- Write a class that models an integer counter where the internal counter variable is initialised to a number entered in the Tester Class when an object of that type was created.

Default & User Defined Constructor

- Counter class with both default and user defined constructors

1.

```
class Counter {

    private int count;

    //default constructor
    Counter(){
        count = 0;
    }

    //user-defined constructor
    Counter(int val){
        count = val;
    }

    public void setCount(int val){
        count = val;
    }

    public int getCount(){
        return count;
    }

    public void increment(){
        count++;
    }

    public void decrement(){
        count--;
    }
}
```

User-defined constructor

2. Counter Test Class

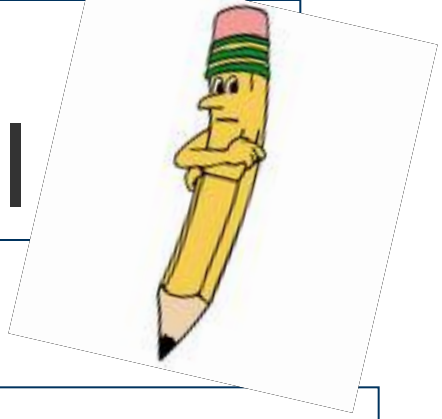
```
class TestCounter {

    public static void main(String[] args){

        Counter c1 = new Counter(10);
        Counter c2 = new Counter(20);

        System.out.println("Values after construction:");
        System.out.println("Counter 1:  "+c1.getCount());
        System.out.println("Counter 2:  "+c2.getCount());
    }
}
```

Call user-defined constructor

Output

```
Values after construction:
Counter 1:  ____
Counter 2:  ____

Process completed
```

Sharpen your Pencil



- Overloaded constructors mean you have more than _____ constructor in your class.
- Overloaded constructors must have _____ (same / different) parameter lists.
- A constructor must have the _____ (same / different) name as the class, and must not have a _____ type.

Summary

- Created Class and Objects
- Created Tester Classes
- Used Constructors --- default & overloaded
- Definitions for
  - Encapsulation
  - Data Hiding
  - Class, Object, Instance