# Object-Oriented Development (CIS1056-N) Worksheet 08: UML Class Diagrams

## Before You Start

Remember: You are not expected to complete the entire brief within the allotted two hours, but to make a start and continue outside of the class.

## Introduction

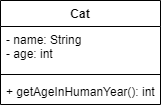
Now we understand the fundamentals of programming, it is time to take a deeper dive into the dominant programming paradigm of our time, object-oriented programming.

Based on the teaching material so far, it should now be apparent that Java is an “object-first” language. You have worked with several Java objects already, notably Scanner and String.

Note Java is not a *purely* object-based language (as Java supports primitive types), but as we need some class structure to write a meaningful program, objects are inescapable.

**Hint:** In all that follows, each class is to be in its own file with the same name as the class name.

## 1. Cat Class

See the rough UML diagram below

1. Your first task is to convert that to a Java class and then provide the implementation of the methods specified.
2. There's no definitive way of converting a cat's age to human years, cats age faster in the first two years of life. According to this [site](https://www.almanac.com/content/cat-age-chart-cat-years-human-years) a 1- year old cat is 15 in human years, a 2 year old cat would be 24 in human years. Each additional year is equivalent to 4 human years.

|  |  |
| --- | --- |
| Cat age | Human age |
| 1 | **15** |
| 2 | **24** |
| 3 | **28** |
| ... | **...** |
| 10 | **56** |

1. Add a new method with the signature:  
     
   **public String meet(Cat otherCat)**

This method will return something like:  
  
**Tiddles meets Felix, they ignore each other!**

1. Add a new Boolean instance variable (attribute) to you cat class called 'friendly'. Modify the constructor method of the Cat class to set this value when you create a cat instance. You will also add a Boolean accessor method with signature:  
     
   **public boolean isFriendly()**
2. The ‘**friendly**’ instance variable will be used to change the way two cats meet each other as indicated in the table below:

|  |  |  |
| --- | --- | --- |
| Cat A | Cat B | Outcome |
| Not friendly | Not friendly | The cats fight! |
| Not friendly | Friendly | Cat A hisses at Cat B! |
| Friendly | Not friendly | Cat A gets hissed at by Cat A! |
| Friendly | Friendly | Cat A meets Cat B, they ignore each other! |

Examples of two cats being created and then meeting:

**Cat catA = new Cat("Tiddles", true);**

**Cat catB = new Cat("Felix", false);**

**catA.meet(catB);**

The output would look something like this:

**Tiddles gets hissed at by Felix!**

## 2. Clock

A clock has an internal mechanism which keeps time as seconds past midnight. The value of seconds can vary between 0 and 86399.

On paper (or use an online tool such <https://app.diagrams.net/>) design new class called Clock. It will have an integer attribute called seconds to represent the number of seconds past midnight.

For example, if seconds contains the value 3672 it represents the time 1:1:12. When seconds is 51303 the time is 14:15:3

The class diagram should include constructors and methods:

1. No parameter constructor to initialise seconds to zero.
2. getSeconds method.
3. setSeconds method.
4. Constructor that initialises the second based on the integer value passed in as a parameter.
5. getTime method that returns the time in the format 13:1:12 (hours:minutes:seconds).  
     
   Write down the pseudocode for how the algorithm to convert the seconds to a string in the format HH:MM:SS.

## 3. Television

Design, create and test a new class called Television. The class diagram is given below:

|  |
| --- |
| Television |
| - channel: int  - volume: int  - power: boolean |
| + Television(int c, int v, boolean p)  + getChannel(): int  + getVolume(): int  + getPower(): boolean  + setChannel(int c): void  + setVolume(int v): void  + setPower(boolean p): void |

## 4. Television Recorder UML class Diagram

Using the the Television UML as a guide, on paper create a UML called TelevisionRecorder.

The TelevisionRecorder class will incorporate a single channel recorder. The record must be able to perform:

* an instant record (start recording the current channel until stop record is executed).
* a scheduled record of the current channel, from a start time to a specified end time.

Update the attributes and add in additional methods to add instant record functionality.

## 5. Television Recorder Class Implementation

Ensure that the methods incorporate appropriate checks to prevent the user performing inappropriate action (e.g. change channel when the recorder is running).

Update your test program to check the new functionality in TelevisionRecorder.

**Dates and Time handling.**

The Java standard library includes a class called LocalDateTime that can do useful things with dates and time!

Example use:

**LocalDateTime inTheFuture = LocalDateTime.now().plusSeconds(5);**

**// ... later ...**

**if(LocalDateTime.now().isAfter(inTheFuture)) {**

**System.out.println("We've reached the future!");**

**}**

As always, familiarise yourself with the documentation: <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/time/LocalDateTime.html>

## 6. Door Controller

A door has an intelligent door controller that manages access to prevent too many people entering the room. It keeps track of the number of people that enter the room and the number that leave. When the room is at capacity, people trying to enter the room will be prevented from doing so. People leaving the room can do so at any time.

1. Paper exercise: Draw the UML class diagram for the Door Controller. Identify the required attributes and behaviours.
2. Create a Java application implementing the class you have designed in 6 (1). Write a test harness to test several instances of the class.

## 7. Vending Machine Simulator

Vending Machine Simulator is an exercise you have been working on in a previous worksheet.

### From Worksheet 4

A vending machine has the following items:

1. Coke (45p)
2. Crisp (35p)
3. Small Chocolate Bar (23p)
4. Bottled Water (32p)

The user is shown a menu and asked to select an item (1-4). Once a valid option has been selected the price is displayed and the user is asked to enter the appropriate amount. The amount is checked, and the appropriate item is dispensed with the correct change. The machine only accepts and returns 1p, 2p, 5p, 10p and 20p coins.

The program is repeated until the shutdown code (999) is entered at the menu prompt.

Write the pseudocode to implement the above operation. include the logic to determine how many of each coin will be required to make-up the change. Show your pseudocode to the tutor and once approved, implement using a new NetBeans project.

### Worksheet 8 Task

Think carefully about the interface, that is how users of the vending machine interact (use) the vending machine. Design a UML class diagram for the vending machine, including its attributes and methods (including methods that may not be public, but provide a service to public methods).

Once you’re satisfied with the design of the class, create a Java application and test it. Create several instances of your class.

## 8. Robot Wars

A robot has hit points, strength and battery power.

Each robot exists to destroy other robots. When they attack it inflicts a random amount of damage of up to and including their strength value. Each attack consumes battery power based on the following table:

|  |  |
| --- | --- |
| Attack % of Strength | Battery Used |
| 0 | **0** |
| <= 25% | **1** |
| <= 50% | **2** |
| <= 75% | **3** |
| <= 100% | **4** |

If a robot’s hit points falls to 0 (or below) the robot is destroyed and sent to the scrap yard. If a robot’s battery power falls below 25% their attack damage is reduced by 50%. When attacking there is a 10% chance that the attempt to attack will fail due a kernel panic in the attacking robot. There is a 5% chance that the robot being attacked will evade the attack.

1. Design the Robot class, with its attributes and methods.
2. Implement the Robot class, writing a test application where two robots will slug it out until one or both of them is sent off to the scrap heap.
3. Have three or more robots fight. In each round, a robot would pick a random other robot to attack. Again, they slug it out until there is one robot (or none) left standing.

## Document History

Revision 0 (07-Nov-21): This is the initial version of the 2022/23 exercise.