



# Java Programming

CPT111 – Week 10  
Erick Purwanto



Xi'an Jiaotong-Liverpool University

西交利物浦大學

CPT111 Java Programming

Week 10 Exercise and Coursework 1

# Polymorphism and Exceptions

# Coding and Submission

---

- Coding in your NetBeans
  - Start with the skeleton code given in the course LMO
  - We are continuing from previous weeks, complete the skeleton code with your code or standard solutions from the previous weeks' exercises' and CW1s' solution
- Submitting into Learning Mall Quiz
  - Do not submit the whole class
  - Only submit the constructor or the method
    - read carefully the instructions
  - You can submit your own private helper method
    - but do not add another public methods

## Exercise Week #10 – Exceptional Clock

---



- Just like in Week 7, complete a class `Clock` that represents time on a 24-hour clock, such as `00:00`, `15:30`, or `23:59`
  - Time is measured in hours (`00 – 23`) and minutes (`00 – 59`)
  - Times are ordered from `00:00` (earliest) to `23:59` (latest)
  - A skeleton file `Clock.java` is given
- In addition, throw an *`IllegalArgumentException`* whenever the argument passed by the client to the constructor or the method is not adhering to the specification of the constructor or the method

## Exercise #10.1 Clock Constructor 1

---

- Complete the first constructor of the class Clock
- It takes two arguments: h and m
  - and creates a new clock object whose initial time is h hours and m minutes
- Throws an *IllegalArgumentException* if either hours is not between 0 and 23, or minutes not between 0 and 59

- Test cases:

```
Clock clock1 = new Clock(1, 0);
```

```
System.out.println(clock1);
```

→ 01:00

```
try {
```

```
    Clock clock2 = new Clock(50, 0);
```

```
} catch (IllegalArgumentException e) {
```

```
    System.out.println("Invalid argument in constructor 1!");
```

```
}
```

→ Invalid argument in constructor 1!

## Exercise #10.2 Clock Constructor 2

---

- Complete the second constructor of the class Clock
- It takes one string argument: s
  - s is composed of two digits, followed by a colon, followed by two digits, so the format is HH:MM such as 02:30
  - it creates a new clock object whose initial time is HH hours and MM minutes
- Throws an *IllegalArgumentException* if either the string argument is not in this format, or if it does not correspond to a valid time between 00:00 and 23:59

- Test cases:

```
Clock clock3 = new Clock("02:30");
```

```
System.out.println(clock3);
```

→ 02:30

```
try {
```

```
    Clock clock4 = new Clock("50:00");
```

```
} catch (IllegalArgumentException e) {
```

```
    System.out.println("Invalid argument in constructor 2!");
```

```
}
```

→ Invalid argument in constructor 2!

## Exercise #10.3 Clock tock

---

- Complete the method tock of the class Clock
- It adds delta minute(s) to the time on this clock, where delta is a positive integer
  - for example, 100 minutes after 02:30 is 04:10
- Note that must **not** use the method in CW1 #7.5 tick()
- Throws an *IllegalArgumentException* if delta is negative, and use the message "Illegal negative delta " followed by the negative number

- Test cases:

```
Clock clock5 = new Clock("02:30");
```

```
clock5.tock(100);
```

```
System.out.println(clock2);
```

→ 04:10

```
try {
```

```
    clock5.tock(-50);
```

```
} catch (IllegalArgumentException e) {
```

```
    System.out.println(e.getMessage());
```

→ Illegal negative delta -50

```
}
```

## Exercise #10.4 Polymorphic Tick Function

---

- Complete the tick function of the class Clock
- It takes a Clock object clock
  - and calls the tick method on it
- It is polymorphic because you can pass an object of a subclass of Clock which overrides tick
- Test cases:  
AlarmClock clock6 = new AlarmClock(5, 59, 6, 0);  
Clock.tick(clock6); → Beep beep beep beep!



# CW1 Week #10 – House of Clocks

---

- A HouseOfClocks object has a collection of many clocks and stores all its clock collections in an array of Clock objects `clocksCollection`
- It also records the number of clocks `numClocks` in its collection
  - the minimum number of clocks is 1,
  - and the maximum number of clocks is 24
- There is a method `printClocks` that prints the time of all the clocks in its collection
  - do **not** modify this method



## CW1 #10.1 HouseOfClocks Constructor 1

---

- Complete the first constructor of the class HouseOfClocks
- It creates a house of clocks which has a collection of numClocks clocks
  - numClocks must be between 1 and 24, inclusive; and throws *IllegalArgumentException*, otherwise
  - the first clock must start at 00:00, the second clock at 01:00, and so on
  - for example,
    - if numClocks = 3, it stores 3 clocks starting at 00:00, 01:00, and 02:00
    - if numClocks = 24, it stores 24 clocks where the last one at index 23 starts at 23:00

- Test cases:

```
HouseOfClocks hc1 = new HouseOfClocks(3);
```

```
hc1.printClocks();
```

→ 00:00↵01:00↵02:00

```
try {
```

```
    HouseOfClocks hc2 = new HouseOfClocks(100);
```

```
} catch (IllegalArgumentException e) {
```

```
    System.out.println("Too much clocks to store!");
```

→ Too much clocks to store!

```
}
```

## CW1 #10.2 HouseOfClocks Constructor 2 (1)

---

- Complete the second constructor of the class HouseOfClocks
- It creates a house of clocks which has a collection of four types of clocks in Week 9:
  - nClock, nAlarm, nCuckoo, nHalloween number of Clock, AlarmClock, CuckooClock, and HalloweenClock objects, respectively
  - all number of clock of any types must be between 0 and 24, inclusive; and the total number of clocks must be between 1 and 24, inclusive; and throws *IllegalArgumentException*, otherwise
  - the first clock must start at 00:00, the second clock at 01:00, and so on
  - the alarm time of the AlarmClock objects must be 1 hour after the starting time
  - for example,
    - if nClock = 1, nAlarm = 1, nCuckoo = 1, nHalloween = 1, it stores total of 4 clocks where the first clock at index 0 is a Clock object starts at 00:00, the second clock at index 1 is an AlarmClock object starts at 01:00, the third clock at index 2 is a CuckooClock object starts at 02:00, and the fourth clock at index 3 is a HalloweenClock object starts at 03:00

## CW1 #10.2 HouseOfClocks Constructor 2 (2)

---

- Test cases:

```
HouseOfClocks hc3 = new HouseOfClocks(1, 1, 1, 1);
hc3.printClocks();
```

→ 00:00↵01:00↵02:00↵03:00

```
for (int i = 0; i < 60; i++) {
    hc3.tick(1);
}
```

→ Beep beep beep beep!

```
for (int i = 0; i < 60; i++) {
    hc3.tick(2);
}
```

→ Cuckoo!↵Cuckoo!↵Cuckoo!

```
for (int i = 0; i < 3; i++) {
    hc3.tick(3);
}
```

→ Halloween!

```
try {
    HouseOfClocks hc4 = new HouseOfClocks(1, 1, 1, -1);
} catch (IllegalArgumentException e) {
    System.out.println("No negative arguments!");
}
```

→ No negative arguments!

## CW1 #10.3 HouseOfClocks Polymorphic Tick (1)

---

- Complete the method tick of the class HouseOfClocks
  - Use polymorphism to call the tick method of the clock in the collection at index clockIndex
    - but, throw an *IndexOutOfBoundsException* if the clockIndex is **not** valid
- and use the message "No clock stored at index " followed by the invalid index

- Test cases:

```
HouseOfClocks hc1 = new HouseOfClocks(3);
```

```
hc1.tick(0);
```

```
hc1.tick(0);
```

```
hc1.tick(1);
```

```
hc1.printClocks();
```

→ 00:02↵ 01:01↵ 02:00

```
try {
```

```
    hc1.tick(100);
```

```
} catch (IndexOutOfBoundsException e) {
```

```
    System.out.println(e.getMessage());
```

→ No clock stored at index 100

```
}
```

## CW1 #10.3 HouseOfClocks Polymorphic Tick (2)

---

- Test cases:

```
HouseOfClocks hc3 = new HouseOfClocks(1, 1, 1, 1);
```

```
hc3.printClocks();
```

→ 00:00↵01:00↵02:00↵03:00

```
for (int i = 0; i < 60; i++) {
```

```
    hc3.tick(1);
```

→ Beep beep beep beep!

```
}
```

```
for (int i = 0; i < 60; i++) {
```

```
    hc3.tick(2);
```

→ Cuckoo!↵Cuckoo!↵Cuckoo!

```
}
```

```
for (int i = 0; i < 3; i++) {
```

```
    hc3.tick(3);
```

→ Halloween!

```
}
```

# Thank you for your attention!

---

- This is the end of Week 10 Exercise and Coursework 1 Task Sheet