# COMP2511

## Tute03

# Agenda

- Design By Contract
- Domain Modelling
- Wondrous

# What is Design By Contract?


# Design by Contract in Assignment i

# Preconditions, Postconditions, Invariants

## PRECONDITION

What is a precondition?

## POSTCONDITION

What is a postcondition?

## INVARIANT

What is an invariant?

# Preconditions, Postconditions, Invariants

## PRECONDITION

Conditions on inputs which guarantee that the postconditions will be true

## POSTCONDITION

Guarantees from the actual software on what you can expect from a function

## INVARIANT

Guarantees from the software that are always maintained before and after a function call

# Liskov Substitution Principle

Objects of a superclass should be replaceable with objects of its subclasses without affecting the correctness of the program.

- Can replace superclass with subclass
- MUST MAINTAIN ALL GUARANTEES!!!

```java
public class Bird {
    private int height = 0;

    /**
     * Make the bird fly to given height.
     * @pre height > 5
     * @param height
     * @post bird will now be flying at height
     */
    public void fly(int height) {
        this.height = height;
    }
}

public class Penguin extends Bird {
    /**
     * Make the penguin fly to given height.
     * @pre height = 0
     * @param height
     * @post nothing changes
     */
    @Override
    public void fly(int height) {
        return;
    }
}
```

# When Penguins Fly?

Penguin overrides Bird's fly method:

- strengthening preconditions (height > 5 => height = 0)
- weakening postconditions (make Bird fly => do nothing to Penguin)

Why does this violate good inheritance design?


# People & Person.java

# Unit Tests & Preconditions?

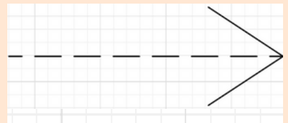
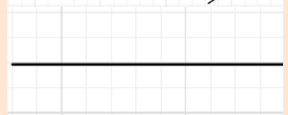
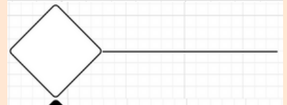
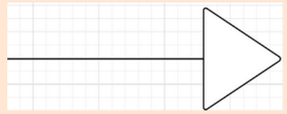
Do we need to write unit tests for cases where preconditions aren’t met?

# YOU SHALL NOT PASS!!!

Let’s try to make our code more defensive by throwing an exception on inputs not satisfying preconditions.

- Are these exceptions considered defined behaviour or not?
- Do we need to account for them in our preconditions?

# UML 101

-  Dependency: source class depends on target
-  Association: source class uses target
-  Aggregation: source class contains target
- Composition: target cannot exist outside of source class - examples?
-  Inheritance: source class is subclass of target
- <<Interface>>
- ***Abstract Method or Abstract Class***


# Domain Modelling


# Wondrous

# IllegalArgument Exception?

Why don’t we need to update the method signature and existing tests?