# Welcome to CS2030 Lab 4!

18 February 2022 [10J]

Only scan if you're physically present!

## Admin Matters

Please login to the PE node first!

```
ssh plabXXXX@peXXX
```

## GitHub Username

Please provide your GitHub usernames to your tutors so that we can track your contributions :)

## Mid Semester Tutor Feedback

A survey has been released on LumiNUS for you to provide teaching feedback about the module so far

Responses are **anonymous**

Please fill it in so that we can continue to improve on the teaching materials :)

## PA1 - Alternative Arrangements

PA1 falls on the Friday of Week 7 (**04/03/2022**)

Please let us know if you have to take tests immediately **before** or **after** your CS2030 lab (e.g. if you have lab from 1000-1200 and your test is at 1400 then it's not counted)

We will book venues for you to take your test **if you need them** (PA1 will end around 10-15 minutes before the end of your lab slot so you can go elsewhere to take the test if you prefer)

Lab TAs to fill in the Google Sheet for affected students (link sent to the Telegram group)

## PA1 - Overview

The practical assessment will last 100 minutes

You will be coding in the PE node (the server that you SSH into every week)

Important: Please **plan** and **think** about your program's design before writing any code!

Full marks if you finish the PE within the assessment period **and** pass all hidden test cases on CodeCrunch

## PA1 - Overview

Moderation period of around one week to modify your code for correctness; the less the modifications, the higher the final moderated score

Style is **not** graded for the PA

Topic coverage: PA1 will cover topics tested until lab 4 (today's lab)

Students must upload a valid FET result **before 8am on Friday**, **4th March**, before being allowed to sit for PA1

## PA1 - Restrictions

You may only use **one device** (i.e. your laptop; no other external devices such as iPads)

You may refer to soft or hard copy notes

You can only use the internet to refer to the Java API (no other websites/resources, including CodeCrunch)

CodeCrunch will be disabled at **10am on the day of PA1** (please remember to download your files the day before if necessary; we will not make any concessions for this)

## Lab 3 Recap - Enhanced For-Loops

Another valid looping mechanism if you just want to do something with each item in a collection of data (the collection must implement the `Iterable` interface; note that this also works with arrays)

```
for (Cruise cruise : cruises) {
    // do something with cruise
}
```

## Lab 3 Recap - Common Code In If-Else Blocks

```
// Notice that Code C is present in both the if and else blocks
if (someCondition) {
    Code A
    Code C
} else {
    Code B
    Code C
}
```

## Lab 3 Recap - Common Code In If-Else Blocks

```
// Better (i.e. move Code C out)
if (someCondition) {
    Code A
} else {
    Code B
}

Code C
```

## Lab 3 Recap - Avoid Multiple Sources Of Truth

```
private final int identifier;

private final Cruise cruise;

RecycledLoader(int identifier, Cruise cruise) {

    super(identifier, cruise);

    // identifier and cruise stored here again

    this.identifier = identifier;

    this.cruise = cruise;

}
```

## Lab 3 Recap - Avoid Multiple Sources Of Truth

```
// Better
RecycledLoader(int identifier, Cruise cruise) {
    super(identifier, cruise);
}
```

Having multiple sources of truth is very unsafe and could lead to bugs if their values ever go out of sync for any reason

If you find that both your superclass and subclass require the same attributes to be stored independently of each other, then you may want to rethink your code design

## Lab 3 Recap - Combining Boolean Expressions In Loop Conditions

Consider the following code sample

```
boolean hasEnded = false;

for (int i = 0; i < 10; i++) {
    // Other code
    if (someCondition) {
        hasEnded = true;
        break;
    }
}

if (hasEnded) { // Do something }
```

## Lab 3 Recap - Combining Boolean Expressions In Loop Conditions

An alternative way to express the same code (move `hasEnded` into the loop condition):

```
boolean hasEnded = false;

for (int i = 0; i < 10 && !hasEnded; i++) {
    // Other code
    if (someCondition) {
        hasEnded = true;
    }
}

if (hasEnded) { // Do something }
```

# Lab 3 Recap - Another Use Case For Static Methods

```
class A {
    private final int num;
    A(int num) {
        this.num = num;
    }
}

class B extends A {
    B(int num) {
        int newNum = ...; // do something to num
        super(newNum);
    }
}
```

What happens when we try to compile the above code?

## Lab 3 Recap - Another Use Case For Static Methods

```
// A better approach
class B extends A {
    B(int num) {
        super(doSomething(num));
    }

    private static int doSomething(int num) {
        return ...;
    }
}
```

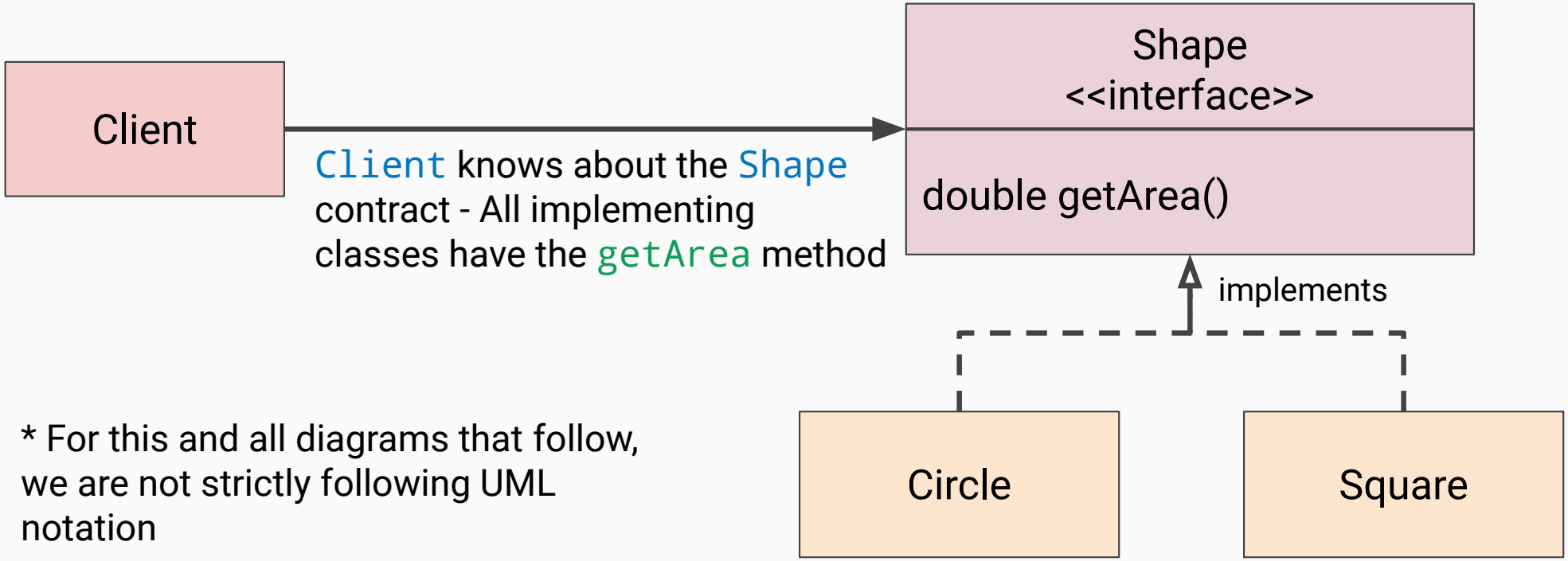
# Error Message Exercise

Tutors please use some examples of error messages that your students have sent you along with the associated code and get your students to debug the code in class

The trickier the bug the better :)

## Interfaces

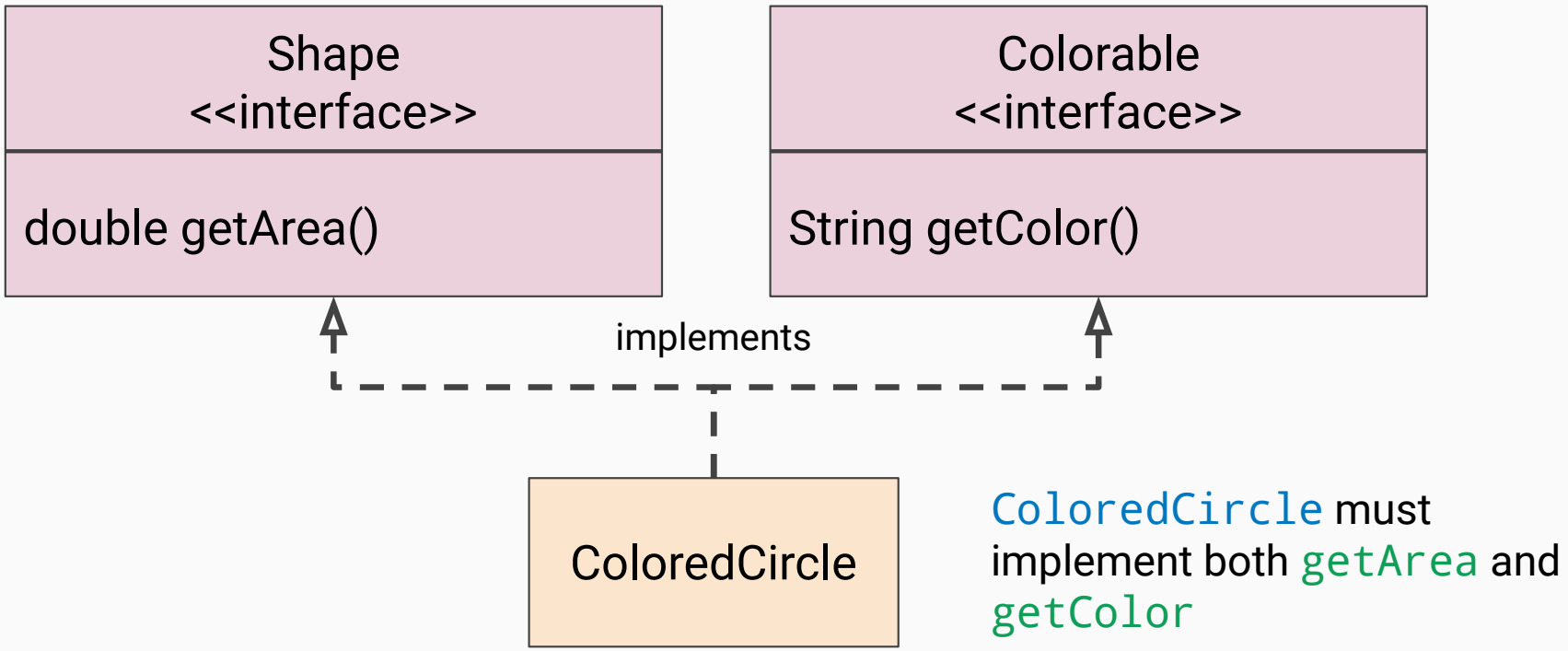
Interfaces provide a contract which specifies some form of behaviour (usually in the form of methods) that implementing objects must conform to




* For this and all diagrams that follow, we are not strictly following UML notation

# Interfaces

Java allows for multiple inheritance through ***interfaces***



## Interfaces

Interface methods are `public` and `abstract` by default (do not need to use those two keywords when declaring interface methods)

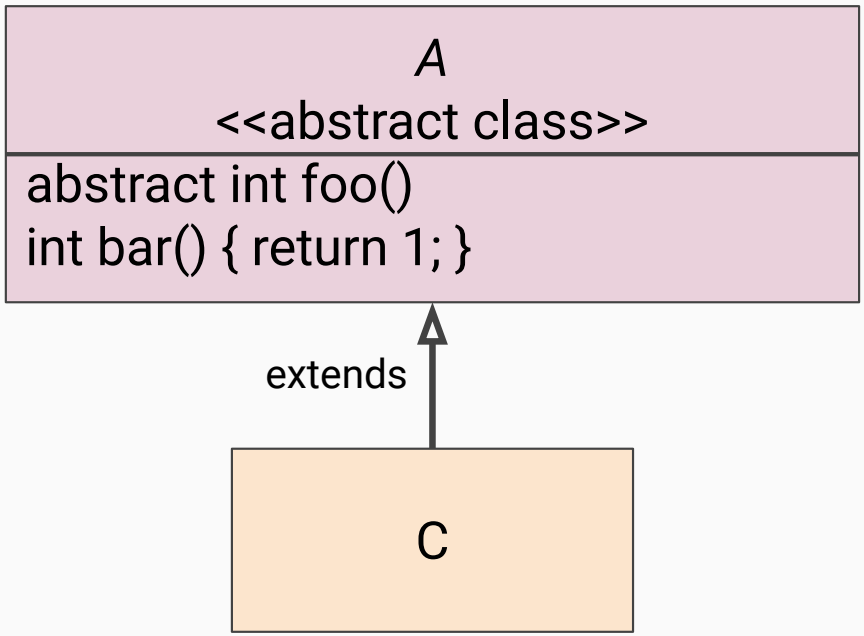
Can use the `default` keyword to provide concrete implementations of methods in interfaces (FYI only but do NOT use this in CS2030)

## Abstract Classes

Abstract classes also specify some form of behaviour subclasses must conform to

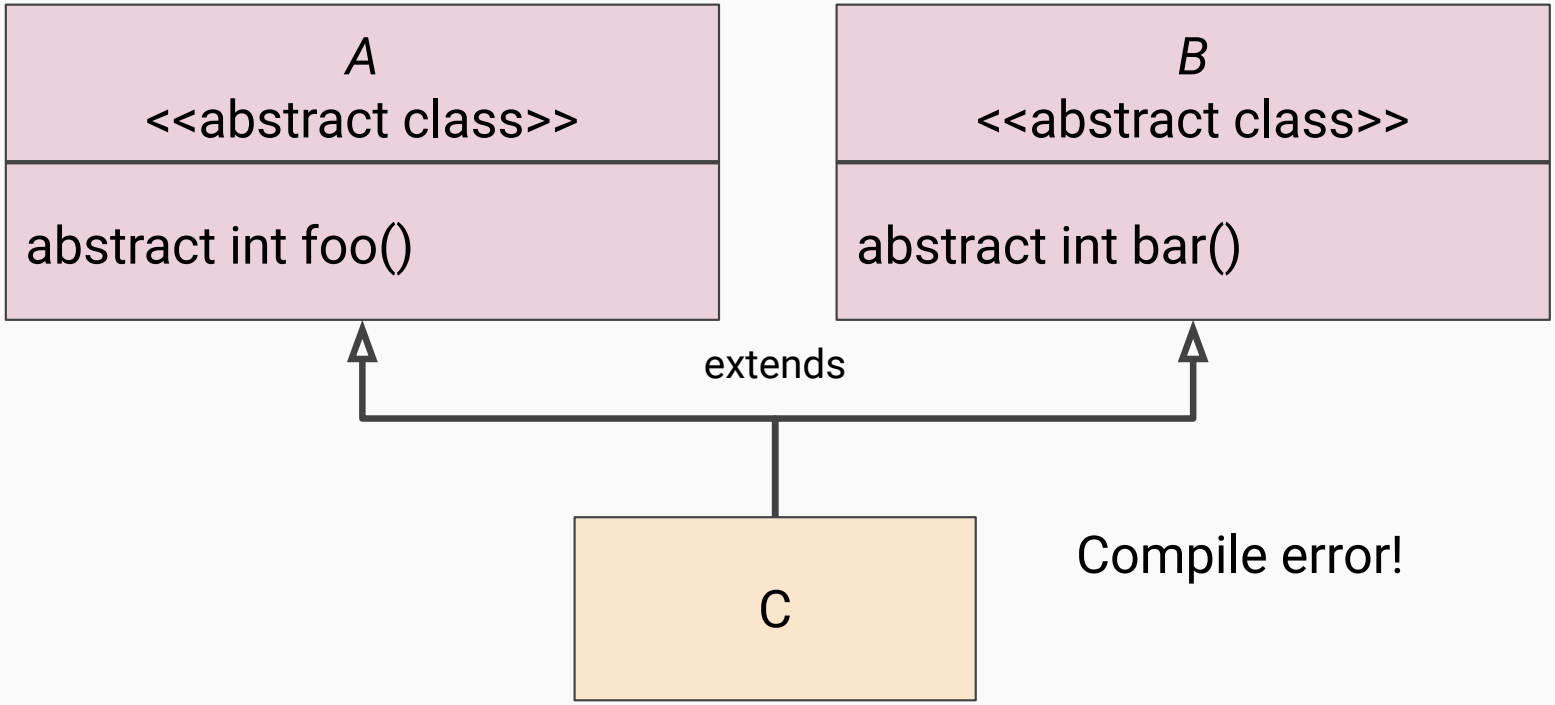
Methods in abstract classes are concrete by default (must provide an implementation inside the abstract class itself)

`A` provides a concrete `bar` method and an abstract `foo` method which concrete subclasses must implement



## Abstract Classes

Java does not allow for multiple inheritance through classes, so subclasses can only extend from one class



## Order of Declaration (Interfaces and SuperClasses)

```
public class A extends B implements C, D, E {

    // Code

}
```

Superclass followed by interface(s). You can remember the ordering by using the keywords in alphabetical order (`extends` before `implements`)

Interfaces are separated with commas

Use the `@Override` tag to ensure that you are overriding/implementing methods properly (compile error if there are mistakes in the code/not actually overriding a parent/interface method)

## Lab 4 Walkthrough (Level 1)

Your tutors will go through level 1 with you towards the end of the session

Please feel free to ask questions/suggest implementations!

For now, we have quite a lot of time remaining so try to treat the rest of the session as practice for PA1 :)

| | Variables | Constructors | Methods |
|---|---|---|---|
| Abstract class | No restrictions. | Constructors are invoked by subclasses through constructor chaining. An abstract class cannot be instantiated using the `new` operator. | No restrictions. |
| Interface | All variables must be `public static final`. | **No constructors.** An interface cannot be instantiated using the `new` operator. | May contain **public abstract** instance methods, **public default** and **public static** methods. |

# Interfaces vs Abstract Classes

- Both can be used to specify common behaviour between objects. How do you decide which to use?
- In general, a strong is-a relationship that clearly describes parent-child relationship should be modelled using classes.

```
class Orange extends Fruit {...}
```

- A weak is-a relationship (is-kind-of relationship) indicates that an object has a certain property, and therefore should be modelled using interfaces.

```
class String implements Comparable {...}
```

```
class Circle extends GeometricObject implements Comparable {...}
```

# Interesting Points about Abstract Classes

1. An abstract method cannot be contained in a non-abstract class. If a subclass of an abstract superclass does not implement all the abstract methods, the subclass must be defined as abstract. In other words, in a non-abstract subclass extended from an abstract class, all the abstract methods must be implemented.
2. Abstract methods are not static.
3. It is possible to define an abstract class that doesn't contain any abstract methods. This abstract class is used as a base class for defining subclasses.
4. A subclass can override a method from its superclass to define it as abstract. This is useful when the implementation of the method in the superclass becomes invalid in the subclass. In this case, the subclass must be defined as abstract.
5. A subclass can be abstract even if its superclass is concrete.

Not actually critical for this lab, but is a core feature of Java.

# The `Comparable<T>` Interface

```
public interface Comparable<T> {
    public int compareTo(T o);
}
```

Many classes in the Java library such as `Integer`, `Double` and `String` already implement it to define a natural order.

```
jshell> new Integer(3).compareTo(new Integer(5));
$1 ===> -1
jshell> "ABC".compareTo("ABC"); // lexicographical order
$2 ===> 0
jshell> new Double(2.1).compareTo(new Double(2));
$3 ===> 1
```

```java
// Compare ages, then names if ages are the same
class Person implements Comparable<Person> {
    int age;
    String name;

    // Other code here

    @Override
    public int compareTo(Person other) {
        if (this.age ≠ other.age) {
            return this.age - other.age;
        }
        return this.name.compareTo(other.name);
    }
}
```

In the above example, we compare names if the ages are the same