Welcome to CS2030S Lab 3!

10 September 2021 [16A]

Lab 2 Recap - Boolean Expressions

```
// Redundant if-else statement
if (booleanMethod(parameter) == true) { // if true == true???
    return true; // return true if true == true
} else {
    return false; // return false if false ≠ true
}

// Better
return booleanMethod(parameter); // Just return the result
```

Lab 2 Recap - if-else Statements

```
if (condition) {
    // some code
} else {
}
```

You do not need to end every `if` statement with an `else` statement.

Using `if` by itself will suffice (e.g. in the following code example)

```
if (condition) {
    // some code
} // do not need else here
```

Lab 2 Recap - Variable Names

👎

```
public class Circle {
    private final double radius;
    public Circle(double r) {
        radius = r;
    }
}
```

👍: Use `this` keyword!

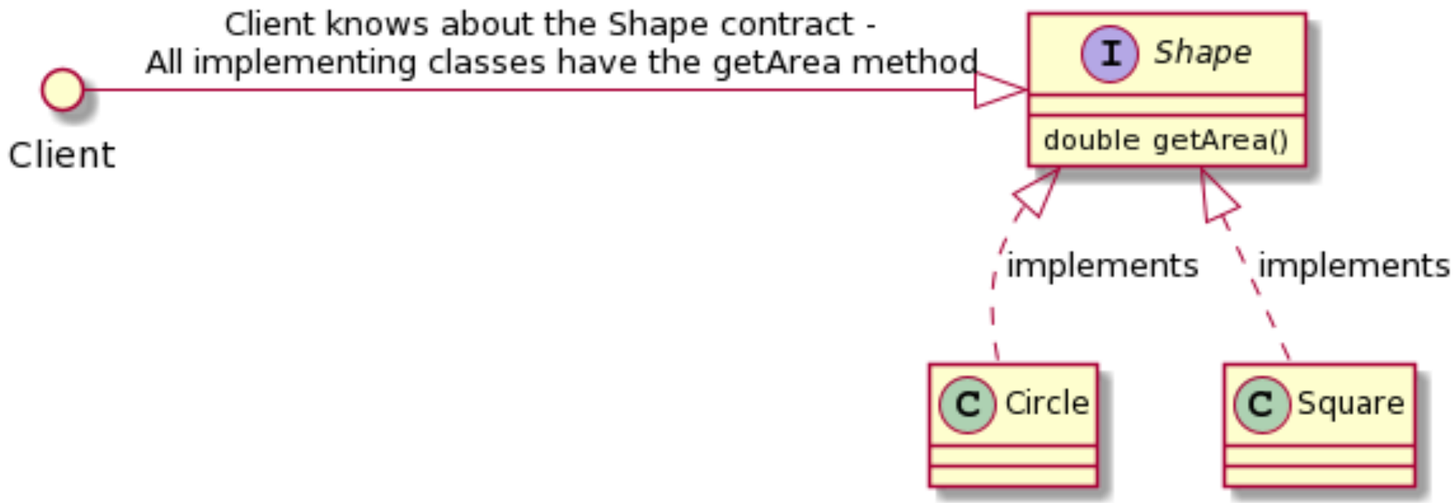
```
public class Circle {
    private final double radius;
    public Circle(double radius) {
        this.radius = radius;
    }
}
```

Lab 2 Recap - Modularisation

- Try to modularise your methods as much as possible.
- As a general rule of thumb, try limiting the length of your methods to 20 or at most 30 lines.
- Exceeding that limit means that your code can probably be shortened/abstracted away

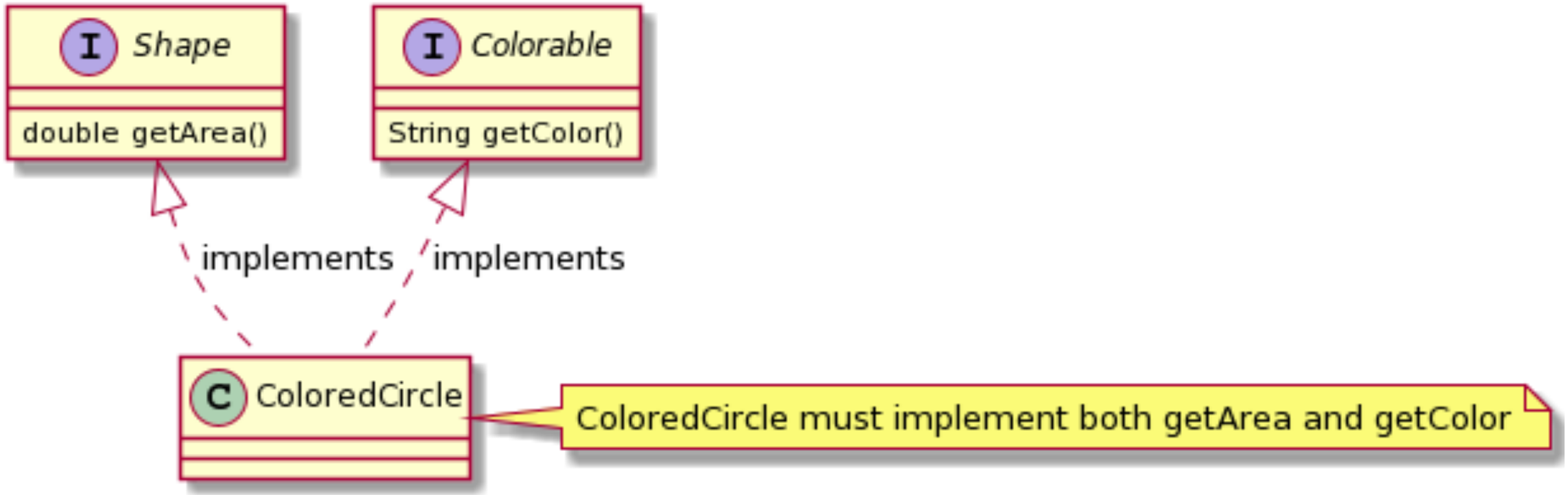
Interfaces

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



Interfaces

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

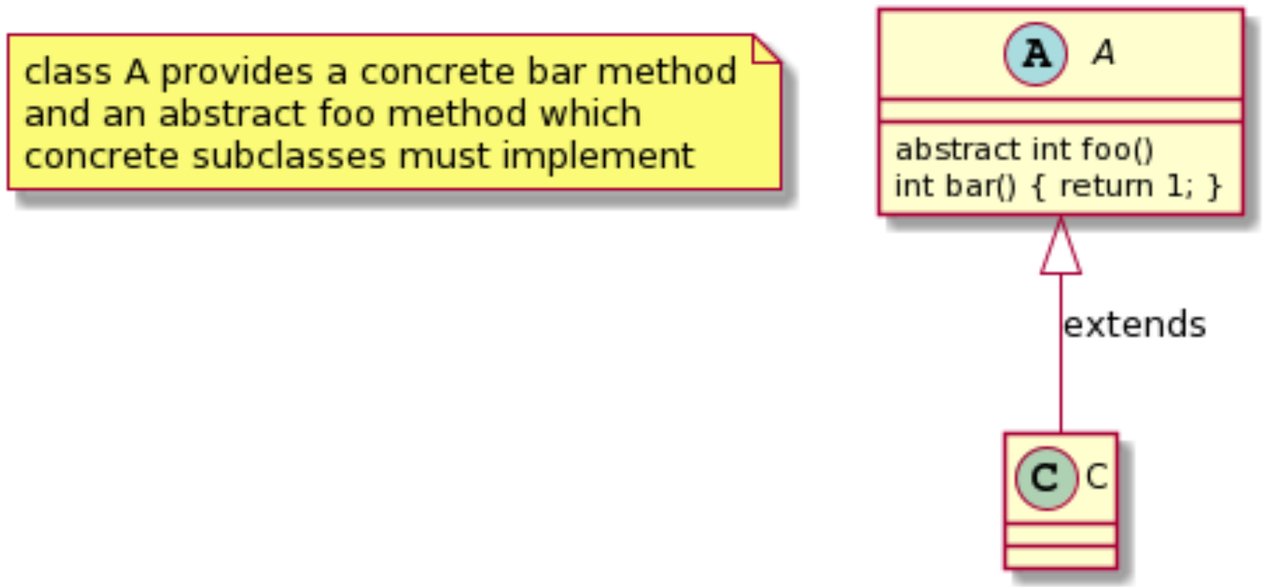


Interfaces

- 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/S)

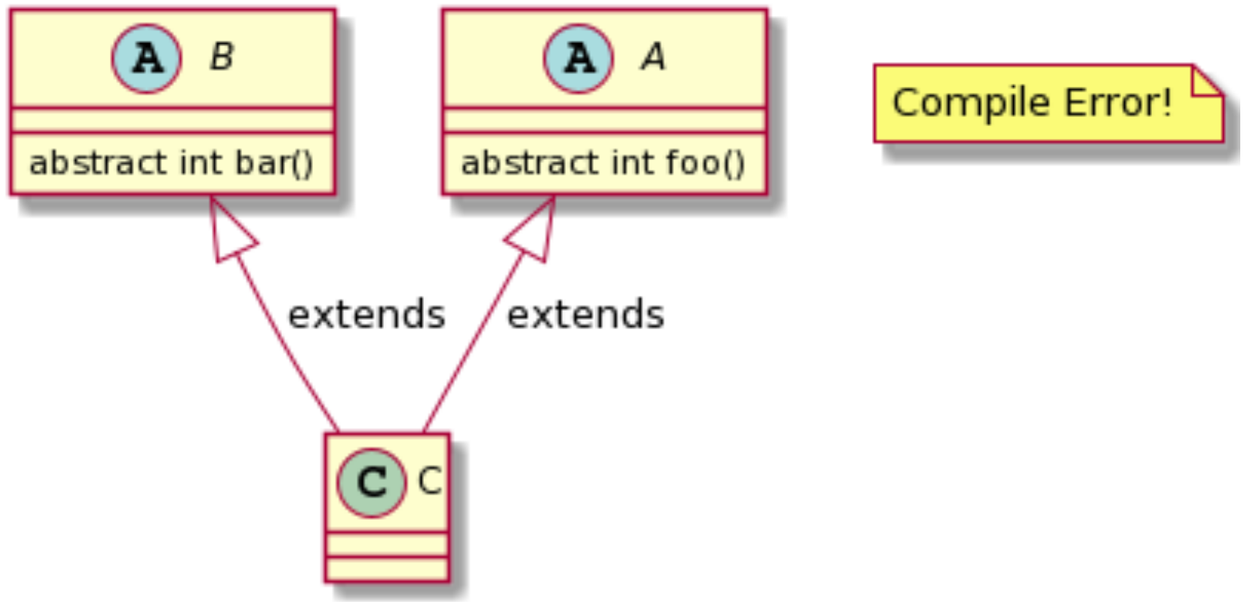
Abstract Classes

- 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)



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 (**e** before **i**).
- Interfaces are separated with commas.
- Use the `@Override` tag to ensure that you are overriding/implementing the methods properly (compile error if there are mistakes in the code/not actually overriding a parent/interface method).

Lab 3: Rubik's Cube

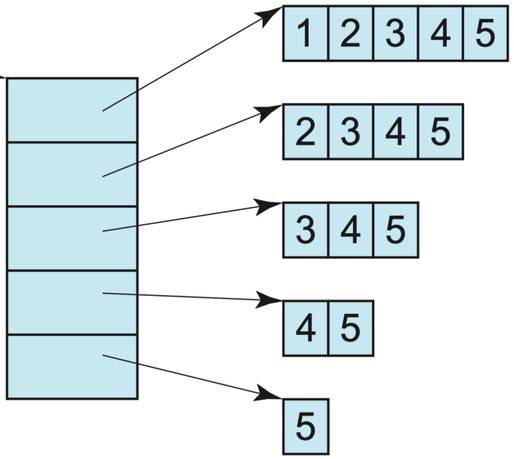
Two-Dimensional Arrays

Syntax:

```
elementType[][] arrayRefVar;
triangleArray[0]; // {1, 2, 3, 4, 5}
triangleArray[0][4]; // 5
triangleArray.length; // 5
triangleArray[3].length; // 2
```

A two-dimensional array is a one-dimensional array in which each element is another one-dimensional array.

```
int[][] triangleArray = {
  {1, 2, 3, 4, 5},
  {2, 3, 4, 5},
  {3, 4, 5},
  {4, 5},
  {5}
};
```



Processing 2D Arrays

Nested `for` loops are often used to process a 2D array.

```
int[][] matrix = new int[10][10];
for (int row = 0; row < matrix.length; row++) {
    for (int column = 0; column < matrix[row].length; column++) {
        // Do stuff to the values
    }
}
```

*A 3D array is an array of 2D arrays... and so on.

Shallow Copy vs Deep Copy

- Shallow copy: object's **reference** is copied rather than its contents.

  ```
  copy1 == copy2 // evaluates to true.
  ```

  - Implications: If you modify `copy1`, you also modify `copy2`.

- Deep copy: all of the object's contents are **duplicated** and have **different memory addresses**.

Visualisation Aid

Please consult us if you're still stuck with the first 2 labs!

	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.