Welcome to CS2030 Lab 3!

11 February 2022 [10J]

Only scan if you're physically present!

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)

Admin Matters

Please login to the PE node first!

```
ssh plabXXXX@peXXX
```

Lab 2 Recap - Classes Over Data Structures

```
// Don't do this!

List<Integer> coordinates = List.of(1, 2);

coordinates.get(0); // What does this value represent?

coordinates.get(1); // What does this value represent?
```

Lab 2 Recap - Classes Over Data Structures

```java
// Better (immediately obvious what the data is for)
class Coordinates {
    private final int row;
    private final int col;

    Coordinates(int row, int col) {
        this.row = row;
        this.col = col;
    }

    // Return statements on the same line to fit code into the slides
    int row() { return this.row; }
    int col() { return this.col; }
}
```

Lab 2 Recap - Classes Over Data Structures

```
// (Continued): Use classes to represent/give meaning cohesive units
of data
Coordinates coordinates = new Coordinates(1, 2);
int row = coordinates.row;
int col = coordinates.col;
```

Lab 2 Recap - Magic Numbers Revisited

```java
// Don't use constants just for the sake of it
private static int HUNDRED = 100; // Not any clearer!
```

Context is important when defining constants

Checkstyle allows for the use of 10 and 100 without declaring constants (e.g. you may need one/both of them for today's lab to extract certain values, but it is also difficult to find names for using them as constants)

Note that -1, 0, 1, and 2 are also allowed to be used without declaring them as constants since they are commonly used for many different purposes

Lab 2 Recap - Ternary Operator

```
int num;

if (someCondition) {
    num = 1;
} else {
    num = 2;
}

// Equivalent
int num = someCondition ? 1 : 2;
```

Lab 2 Recap - Ternary Operator

Intuitively, `a ? b : c` means if expression `a` (a boolean expression) evaluates to `true`, use expression `b`, else use expression `c`.

Known as the ternary operator

Use this construct to shorten some of your code (personal preference)

Lab 2 Recap - Grid Movement

```
if (row - 1 ...) { // move up }
if (row + 1 ...) { // move down }
if (col - 1 ...) { // move left }
if (col + 1 ...) { // move right }
```

What if we wanted to enable diagonal movement? Do we code 8 if-statements? :O

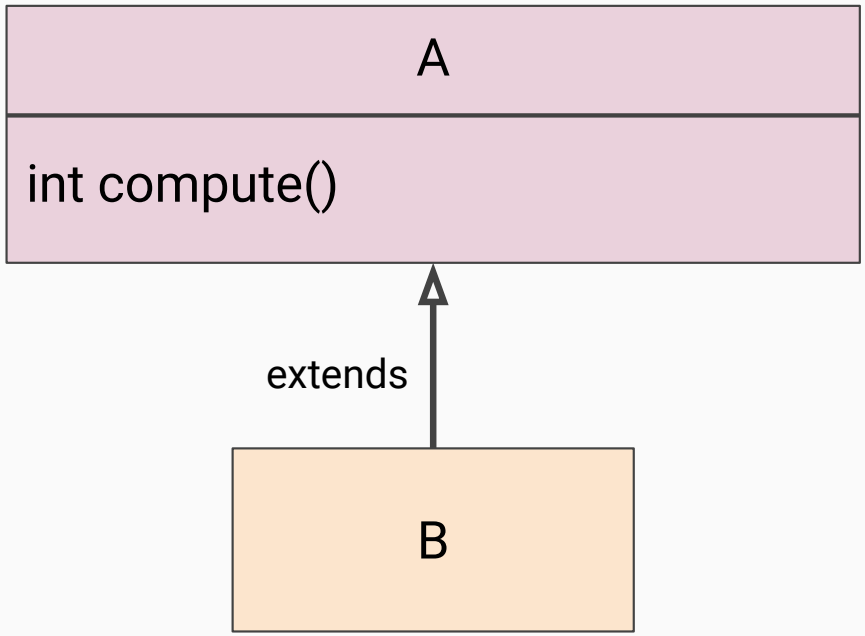
Lab 2 Recap - Grid Movement

```java
// verticalMoves.get(i) and horizontalMoves.get(i) will be paired together
ImList<Integer> verticalMoves = new ImList<>(List.of(-1, 1, 0, 0));
ImList<Integer> horizontalMoves = new ImList<>(List.of(0, 0, -1, 1));

// New move combinations can be added to verticalMoves and horizontalMoves
// without changing the rest of the code
for (int i = 0; i < verticalMoves.size(); i++) {
    int nextRow = row + verticalMoves.get(i);
    int nextCol = col + horizontalMoves.get(i);

    if (boundary checks...) {
        // do something with nextRow and nextCol
    }
}
```

Inheritance/Subclassing



B (by subclassing A) implicitly has access to all non-private attributes and methods in A

Inheritance/Subclassing

```
class A {
    private final int num;

    A(int a) {
        this.num = num;
    }

    int compute() {
        return num + num;
    }
}
```

```
class B extends A {
    @Override
    int compute() {
        // Note that simply calling compute() will
        // recursively call B's compute()
        // and the program will eventually crash
        return super.compute() + super.compute();
    }
}
```

Inheritance/Subclassing

Some example scenarios:

```
A a = new A();
A b = new B();
A c = new C();

a.foo(); // Calls A's foo()
b.foo(); // Calls B's foo()
c.foo(); // Calls C's foo()

b = c; // Valid
a = b; // Valid
a = c; // Valid
```

```
class A {
    int foo() { ... }
}

class B extends A {
    int foo() { ... }
}

class C extends A {
    int foo() { ... }
}
```

Inheritance/Subclassing

Some example scenarios:

```
A a = new A();
A b = new B();
A c = new C();

// The variables b and c are of type A,
// and the methods bar() and baz() don't
exist in A
b.bar(); // Invalid (compile error)
c.baz(); // Invalid(compile error)
```

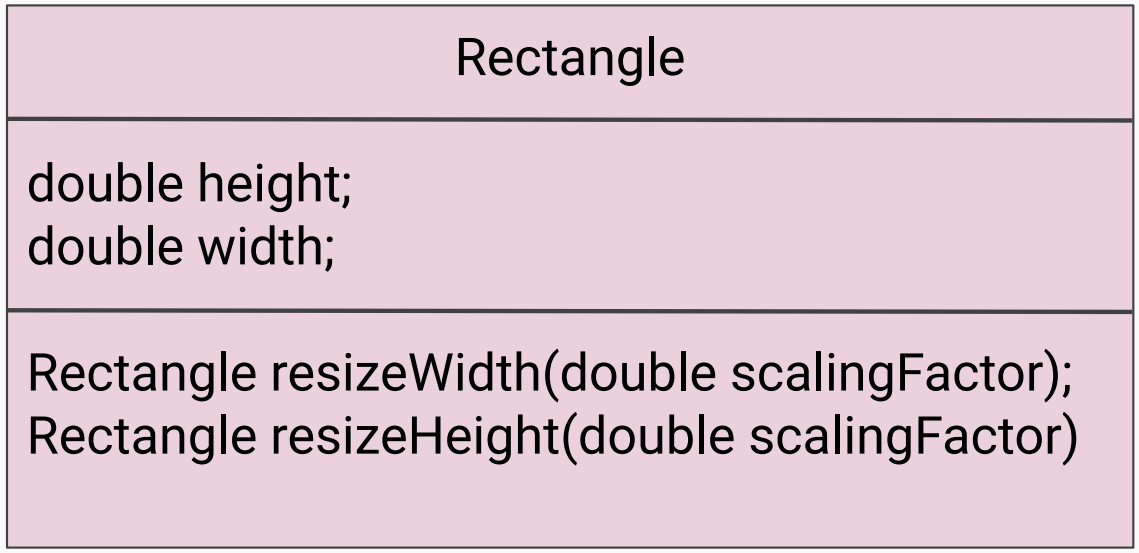
```
class A {
    int foo() { ... }
}

class B extends A {
    int foo() { ... }
    int bar() { ... }
}

class C extends A {
    int foo() { ... }
    int baz() { ... }
}
```

Substitution

Consider the following Rectangle class:



Question: Is a `Square` a `Rectangle` (i.e. should `Square` be a subclass of `Rectangle`)?

Lab 3 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!