Welcome to CS2030S Lab 2!

3 September 2021 [16A]

Admin Matters

1. Sit in front!
2. Make sure you are CRYSTAL CLEAR of how to login to PE server! (**Do it now**)
   How to navigate vim.
3. Telegram chat is our **exclusive** communication channel.
4. Clarify any basic Java concepts! Syntax, keywords, types, primitive, wrapper, english.
5. Trial and error learning. Practice!
6. Any feedback? Complaints? Slides are useless/ugly?

PA1 Alternative Arrangements

- PA1 falls on the Friday of Week 7 (**01/10/2021**)
- Please let us know (Telegram group) if you have to take tests immediately **before** or **after** your CS2030/S 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 1 Common Mistakes

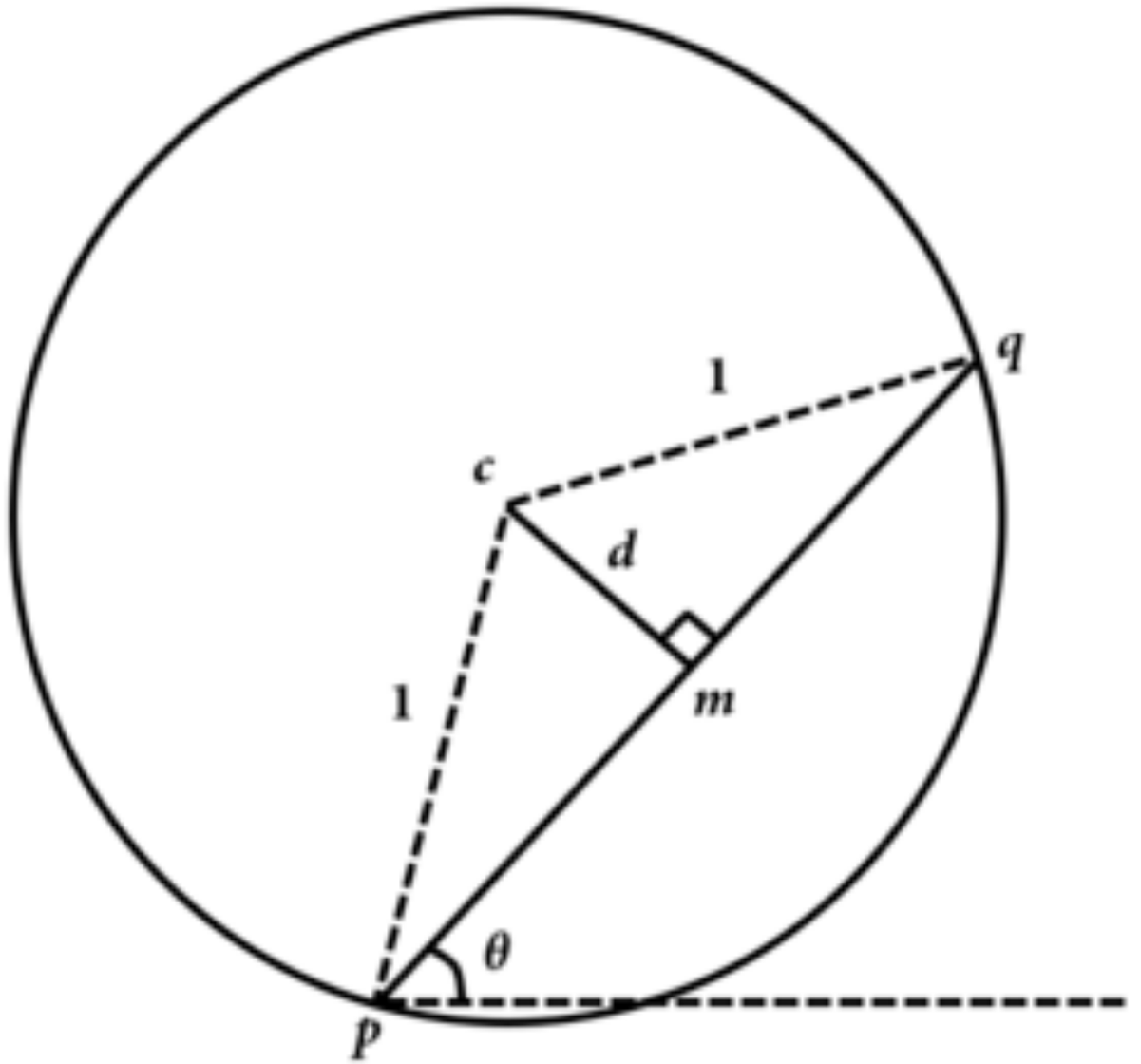
1. Not checking if $d(p, q) \leq 2$ or $d(p, m) \leq 1$.
2. `Point::contains(Point center)` or `Point::contains(Circle c)` — contains method should be **encapsulated** in the `Circle` class instead to reduce coupling.
3. Convoluted code (invoking multiple methods in one line)[1]

```
Circle createUnitCircle(Point p, Point q) {

   Point m = p.midPoint(q);

   return new Circle(m.moveTo(p.angleTo(q) +
Math.PI/2, Math.sqrt(1 -
Math.pow(p.distanceTo(m), 2))), 1.0);

}
```



[1] taken from one of you

Lab 1 Recap - Imports

```
import java.lang.*;
```

Note that these libraries are automatically imported!

As a result, you do not need to import them in your java file.

Lab 1 Recap - Style - Use of Spaces

Use spaces after operators and punctuation marks

Good

```
int x = 1 + 2;
String.format("%s", "hi");
```

Bad

```
int x=1+2;
String.format("%s","hi");
```

Lab 1 Recap - Style - Variable Naming

Use more descriptive variable names

```
public class Circle {
    private final Point p; // Not advisable
    private final Point midpoint; // Better variable name
}
```

Spell out variables in full (e.g. `maxDiscCoverage` instead of `mdc`)

The convention in Java is to use camelCase (`helloWorld` instead of `hello_world`)

Lab 1 Recap - Style - if Statements

Use braces after single line `if` statements

```
// works but not advisable; could result in bugs if not careful
if (condition)
    // some code

if (condition) {
    // code here;
}
```

Lab 1 Recap - Style - Line Wrapping

- Wrap lines instead of letting them get too long!
- CS2030 sets **80 characters** as the line length limit.
- It is usually appropriate to wrap lines **after operators** or at appropriate junctures for Strings (e.g. after a full-stop)

Lab 1 Recap - Style - String.format()

- `String.format()` can be used to format entire strings instead of being called multiple times.
- The following lines return the same `String` (assume that x and y are `int` variables)

```
return "Coordinates: " + String.format("(%d, %d)", x, y);
```

```
return String.format("Coordinates: (%d, %d)", x, y);
```

- Use `%s` as the placeholder for a String

Lab 1 Recap - Style - Variable Initialization

Not recommended

```
int numberOfPoints;
// some other code
numberOfPoints = sc.nextInt();
```

Better — declare and initialize variables within the scope that they are needed

```
// Other code
int numberOfPoints = sc.nextInt(); // Declare and initialize here
```

Lab 1 Recap - Style - Variable Initialization

Not Recommended

```
int i, j, k; // i, j, k all outside the scope of the for loops
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        // Other code
    }
}
```

Better — declare and initialize variables within the scope that they are needed

```
for (int i = 0; i < n; i++) { // i in the scope of this for loop
    for (int j = 0; j < n; j++) { // j in the scope of this for loop
        // Other code
    }
}
```

Lab 1 Recap - Style - Arranging Methods

👎 **Bad**

```
if (!isUnusualCase) {
    if (!isErrorCase) {
        start();
        process();
        cleanup();
        exit();
    } else {
        handleError();
    }
} else {
    handleUnusualCase();
}
```

👍 **Good**

```
if (isUnusualCase) {
    handleUnusualCase();
    return;
}

if (isErrorCase) {
    handleError();
    return;
}

start();
process();
cleanup();
exit();
```

Lab 1 Recap - Style - CS2030 Style Guide

1. Original style guide
2. Github Wiki

Lab 1 Recap - Style - Modularisation

- Break up length methods into shorter ones that ultimately have the same functionality.
- If need be, it is also recommended to abstract out lines of code into helper functions even if it's only being used once.
- This makes your code easier to debug and is an important part of designing code.

Lab 1 Recap - Style - Modularisation

Example of modularisation:

```
boolean containsPoint(Point q) {
    return center.distTo(q) ⩽ radius;
}
```

You can make use of the above method as necessary insteaf of repeatedly calling `centre.distTo(q) ⩽ radius`.

Lab 1 Recap - Style - Data Hiding

- Declare instance and class variable as private (except for special cases like constants).
- This is part of *encapsulation* and hides data that only the class or instance needs to know about.

Lab 1 Recap - Style - Immutability

- Make objects immutable so that they cannot be tampered with; especially important for reference types
- Use the `final` keyword for instance variables and use constructors to get new object instances.
- No setters (e.g. a `void setX(double x)` method)! We do not change the state of immutable objects once they are created.

`static` keyword

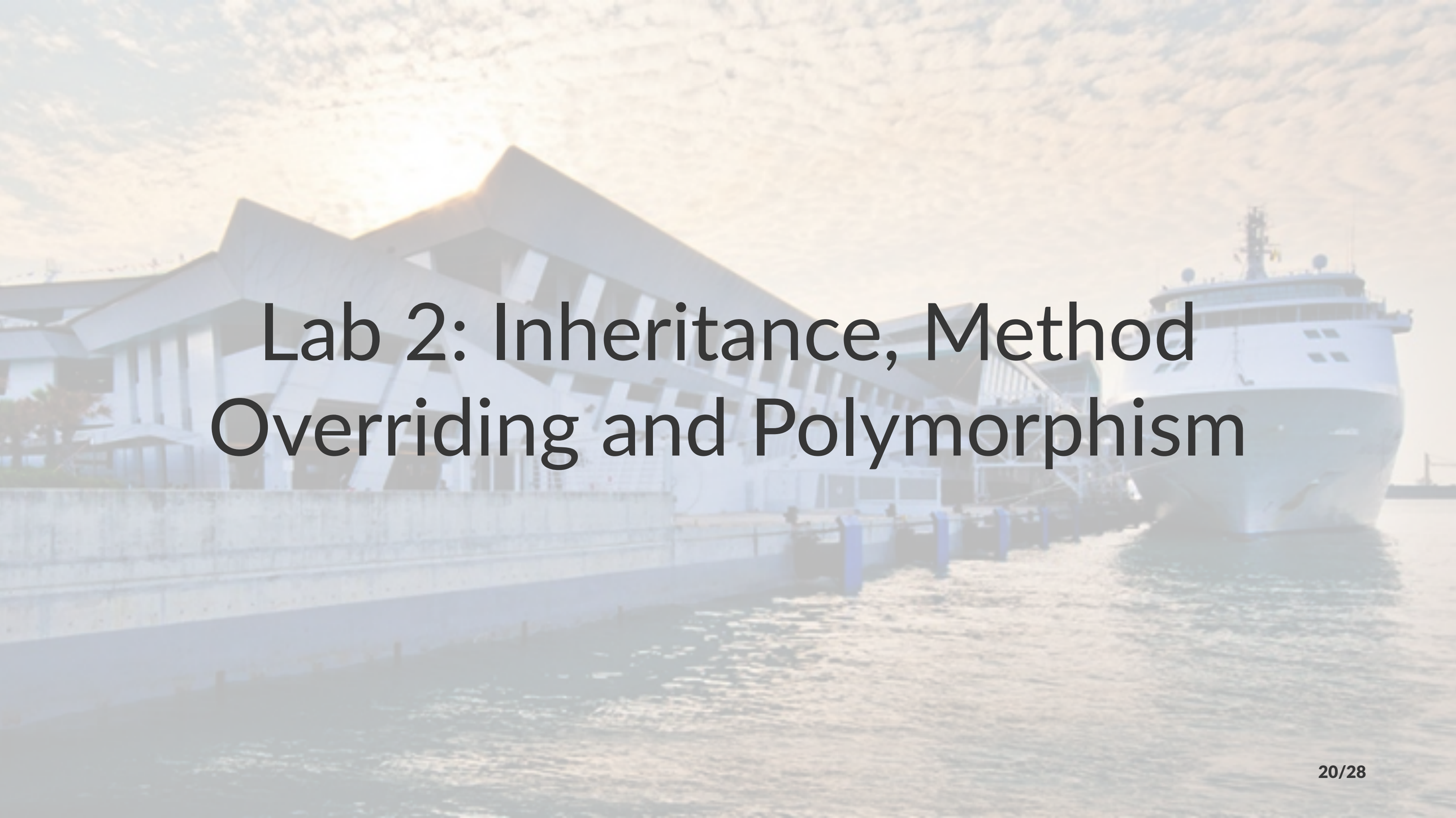
- The `static` keyword is used to declare class level attributes.
- One copy of each static variable (attribute) is stored across all instances of a class (instead of one copy per instance).

```
class Dog {

  private static String sound = "woof";

}
```

- For example, in the above class, the "woof" sound is shared across any instance of a `Dog`.



Lab 2: Inheritance, Method Overriding and Polymorphism

Inheritance

- **Superclasses and Subclasses**
  - Inheritance enables you to define a general class (i.e., a superclass) and later extend it to more specialized classes (i.e., subclasses).
  - is-a relationship
- The keyword `super` refers to the superclass and can be used to invoke the superclass's methods and constructors.

Method Overriding

- To override a method, the method must be defined in the subclass using the same signature as in its superclass.
- **Overloading** means to define multiple methods with the same name but different signatures. **Overriding** means to provide a new implementation for a method in the subclass.
- Every class in Java is descended from the `java.lang.Object` class.
  - `toString()` method

Spot the Difference!

```java
public class Test1 {
    public static void main(String[] args) {
        A a = new A();
        a.p(10);
        a.p(10.0)
    }
}

class B {
    public void p(double i) {
        System.out.println(i * 2);
    }
}

class A extends B {
    public void p (double i) {
        System.out.println(i);
    }
}
```

```java
public class Test2 {
    public static void main(String[] args) {
        A a = new A();
        a.p(10);
        a.p(10.0)
    }
}

class B {
    public void p(double i) {
        System.out.println(i * 2);
    }
}

class A extends B {
    public void p (int i) {
        System.out.println(i);
    }
}
```

Method Overloading

1. Different number of arguments
2. Different types of arguments.
3. Different order of arguments.

✅

```
void eat(Food food, Drink drink);

void eat(Food food);
void eat(Food food, Food snack);
void eat(Drink drink, Food food);
```

❌

```
void eat(Food food);

void eat(Food snacc);
Human eat(Food food);
```

Why use `@Override`?

- This annotation denotes that the annotated method is required to override a method in its superclass.
- If a method with this annotation does not override its superclass's method, the compiler will report an error.
- For example, if toString is mistyped as tostring, a compile error is reported. If the `@Override` annotation isn't used, the compiler won't report an error.

Polymorphism

- Polymorphism means that a variable of a supertype can refer to a subtype object.
- A subclass is a specialization of its superclass; every instance of a subclass is also an instance of its superclass, but not vice versa.
- Every circle is a geometric object, but not every geometric object is a circle.

```
void displayObject(GeometricObject object) {
    System.out.println(String.format("%s %s of dimension %s",
                                     object.getColor(),
                                     object.getShape(),
                                     object.getDimension()));
}

displayObject(new Circle(1, "red"));
displayObject(new Rectangle(1, 1, "black"));
displayObject(new Object()); // a superclass
```

```
[Output]
red circle of dimension 1
black rectangle of dimension 1x1
| Error:
| incompatible types: java.lang.Object cannot be converted to GeometricObject
| displayObject(new Object())
|               ^----------^
```

```
elinks index.html
```

Tips

1. What types of Cruises and Loaders are there?

2. What relationships do they share?