
Australian National University
Structured Programming
COMP1110/6710
Needs ANU Account!
pollev.com/fabianm
Register for Engagement
Image Courtesy NASA/JPL-Caltech.

Admin

- Class representatives – apply by Wednesday COB
- U1 out now (COMP1110)
- P1 due Friday (COMP6710)
- Design Recipe Guide on Course Website
- Functional Java Description On Course Website Extended
- New Standard Library Version: 2025S1-3
- Functional Java Available on EdStem

Recap: Communication

Make sure your questions on Ed Discusson have a meaningful title

- Good: How do I get the current time?
- Bad: Question/Concern/Doubt/etc on/about/regarding/etc ...

If we reject a question because of its form, please post an improved version.

See Expectations on your usage of Ed Discussion

And Who to Contact and How Guide

Sign up for Ed! Course Link in Wattle

Addendum: Saving Files

VSCod(ium/e) does not save your files if you don't tell it to, and so Java/Git won't see your changes.

Press CTRL+S to save your file every time before you run Java or Git commands.

Reminder: Check out MIT's Missing Semester Course: https://missing.csail.mit.edu/

Recap: Evaluation Order

[Live Demo]

I’m going to use IntelliJ here. Don’t use it on your own until Week 6!

Shapes

Design a program that considers various geometric shapes: circles, rectangles, and right triangles.

Design two functions that work on any given shape, and calculate, respectively, the area and circumference of the shape.

The Design Recipe

Step 1. Problem Analysis and Data Design

> Now addeding: General Itemizations

Australian National University

Data Design

So far:

- Basic Data Types
- Enumerations
- Records

General Itemizations

For when you have a limited number of distinct cases, but with is associated with more data.

Consist of:

- A record type for each case
- A *sealed interface* (NEW!) to group the records into an itemization

General Itemizations

For when you have a limited number of distinct cases, but with is associated with more data.

Overall interpretation of data definition

```
/** A Shape represents a geometric shape and is one of:
 *  - A Circle, representing circular shapes
 *  - A Rectangle, representing rectangular shapes
 *  - A RightTriangle, representing right triangles
 */
sealed interface Shape
  permits Circle, Rectangle, RightTriangle {}
```

Cases of the itemization, high-level

Name of Java type for itemization

Java requires that you list the cases explicitly, and define matching records (next steps)

General Itemizations

For when you have a limited number of distinct cases, but with is associated with more data.

Good idea to mention overall type name

```
/** A Circle is a Shape characterized by its radius.
 *  Examples:
 *  - A circle of radius 1.0 ; A circle of radius 0.5
 *  @param radius – radius of the circle, in whatever units
 *                the program uses for Shapes, non-negative */
record Circle(double radius) implements Shape {}
```

The rest are normal record definitions

Java needs reference back to sealed interface

General Itemizations

For when you have a limited number of distinct cases, but with is associated with more data.

Good idea to mention overall type name

```
/** A Rectangle is a Shape defined by its width and height.
 * Examples:
 * - A rectangle of width 1.0 and height 0.5
 * - A rectangle of width 2.0 and height 2.0
 * @param width – …
 * @param height – …*/
record Rectangle(double width, double height) implements Shape {}
```

Java needs reference back to sealed interface

The rest are normal record definitions

General Itemizations

For when you have a limited number of distinct cases, but with is associated with more data.

Good idea to mention overall type name

```
/** A RightTriangle is a Shape defined by the lengths
 * of its two legs.
 * Examples: …
 * @param leftLeg – …
 * @param rightLeg – …*/
record RightTriangle(double leftLeg, double rightLeg)
                     implements Shape {}
```

The rest are normal record definitions

Java needs reference back to sealed interface

General Itemizations - Templates

For general itemizations, the general scheme is:

```
// { …
//    return  … switch(x) {
//      case [Record1](var [rec1field1], …) -> …;
//      […];
//      case [RecordN](var [recNfield1], …) -> …;
//    } …;
// }
```

Key Motto:

The shape of the data determines the shape of the code!

Template Example - Shapes

```
// { ...
//   return ... switch(shape) {
//     case Circle(var radius) ->  ... ;
//     case Rectangle(var width, var height) ->  ... ;
//     case RightTriangle(var leftLeg, var rightLeg) ->  ... ;
//   } ...;
// }
```


Remember COMP1100?

```
data Shape = Circle Num
           | Rectangle Num Num
           | Triangle Num Num

shapeFun :: Shape -> …
shapeFun (Circle r) = …
shapeFun (Rectangle w h) = …
shapeFun (Triangle l r) = …
```


The Design Recipe

Step 2. Function Signature and Purpose Statement

Australian National University

Exercise

Design two functions that work on any given shape, and calculate, respectively, the area and circumference of the shape.

What are the signatures and purpose statements for these?

The Design Recipe

Step 3. Examples

Australian National University

Examples

Now that the signature for your function is clear, come up with some example inputs and the corresponding expected outputs.

Examples should cover all major cases you would expect your function to handle.

Examples should be understandable to a human reader.
Be detailed, but not at the cost of clarity.

Examples – An Example

```
/**
 * Computes the sum of two integers.
 * Examples:
 * given: 5, 2   expect: 7
 * given: -14, -2   expect: -16
 * @param … */
int sum(int x, int y) {}
```

Inputs

Outputs

Wherever possible, write precise values. Easy to do with primitive types and Strings.

Examples – Another Example

```
/**
 * Creates an image representing the state of the world
 * Examples:
 * given: A state in which the player is currently aiming a
 *        bird at some objects
 * expect: An image that shows the respective bird at the
 *         starting position, with aiming lines showing the
 *         current shot angle, and any left-standing objects
 *         in the target area
 * … */
Image draw(State state) {}
```

Precise data would be infeasible here, but you can still write very concrete descriptions

Examples – Yet Another Example

```
record Vector2D(double x, double y) {}
/**
 * Computes the magnitude of a vector
 * Examples:
 * given: (3,4)  expect: 5
 * given: (12,-5)  expect: 13
 * @param … */
double magnitude(Vector2D vec) {}
```

You can still often make reasonably precise representations, even for more complex data

Exercise

Design two functions that work on any given shape, and calculate, respectively, the area and circumference of the shape.

What are examples for these?

The Design Recipe

Step 4. Design Strategy

Australian National University

Recap: 4 Design Strategies

- Simple Expressions
- Combining Functions
- Case Distinction
- Template Application

Exercise

Design two functions that work on any given shape, and calculate, respectively, the area and circumference of the shape.

Which design strategies should we choose?

The Design Recipe

Step 5. Implementation

Australian National University

Time to Write Some Code

[Live Demo]

The Design Recipe

Step 6. Tests

Australian National University

Why Test?

- To gain some confidence that code does what it should
- To guard code against accidental changes
- To guard code against bugs being reintroduced (“regression tests”)

What to Test?

EVERYTHING!

<u>In this course:</u>
All functions that need to follow Design Recipe and whose return type is not `Image`

Image from memegenerator.net via Erik Dietrich
https://daedtech.com/test-readability-best-of-all-worlds/

How to Come Up With Test Cases

First, turn your examples into code.

Then, consider all possible execution paths, including helper functions.

if/switch statements and expressions mean not every piece of code is executed all the time. You want to cover all potential paths and interesting combinations. This is called *test coverage*.

Kinds of Tests

Lots – see COMP2120/6120

We are doing *unit* tests.

Writing Tests

3 levels:

- The `test()` function
- Test cases, each defined in their own function
  - Run them from the `test()` function via `runAsTest`
  - `Write short summary of/rationale for test case as comment above test function`
- Test assertions, one or more in each test case
  - `testEqual`
  - `testNotEqual`
  - `testTrue`
  - `testFalse`

Test Example

```
/** Tests that the sum of 5 and 2 is 7 */
void testSumExample1() {
  testEqual(7, sum(5,2), “5 + 2 should be 7”);
}
```

Equality assertion. By convention, expected value is first argument.

```
void test() {
  runAsTest(this::testSumExample1);
}
```

Run with `java --enable-preview comp1110.testing.Test yourFile.java`

Exercise

Design two functions that work on any given shape, and calculate, respectively, the area and circumference of the shape.

Let’s write some tests!

Designing World Programs

The Universe Package

Australian National University

Images

Each Image has a bounding box, with width and height in pixels

Each Image has a *Pin* location, by default in the center of the bounding box.

Coordinates go from top-left corner rightwards and downwards.

➔ Y-coordinates grow downwards.

Images

Base Images:

- Rectangle
- Circle
- Ellipse
- Polygon
- Text
- Image Files

Fill Mode: Solid vs. Outline

Colour

See library documentation on Functional Java Page.

Images

See library documentation on Functional Java Page.

Rotate/Scale/Combine

Combinations:

- Overlay – put one image on top of another, use pins to align
- OverlayXY – same as above, but put pin of top image at x/y relative to top left image of base
- Place/PlaceXY – like corresponding overlay, but image keeps dimensions of base image (top image may be cut off)

BigBang

Pick data type for world state, e.g. `World`

Pick initial world state, i.e. a value of type `World`

Provide at least:

```
Image draw(World w) {}
```

Optionally:

```
World step(World w) {}
World onKeyEvent(World w, KeyEvent kev) {}
World onMouseEvent(World w, MouseEvent mev) {}
boolean hasEnded(World w) {}
```

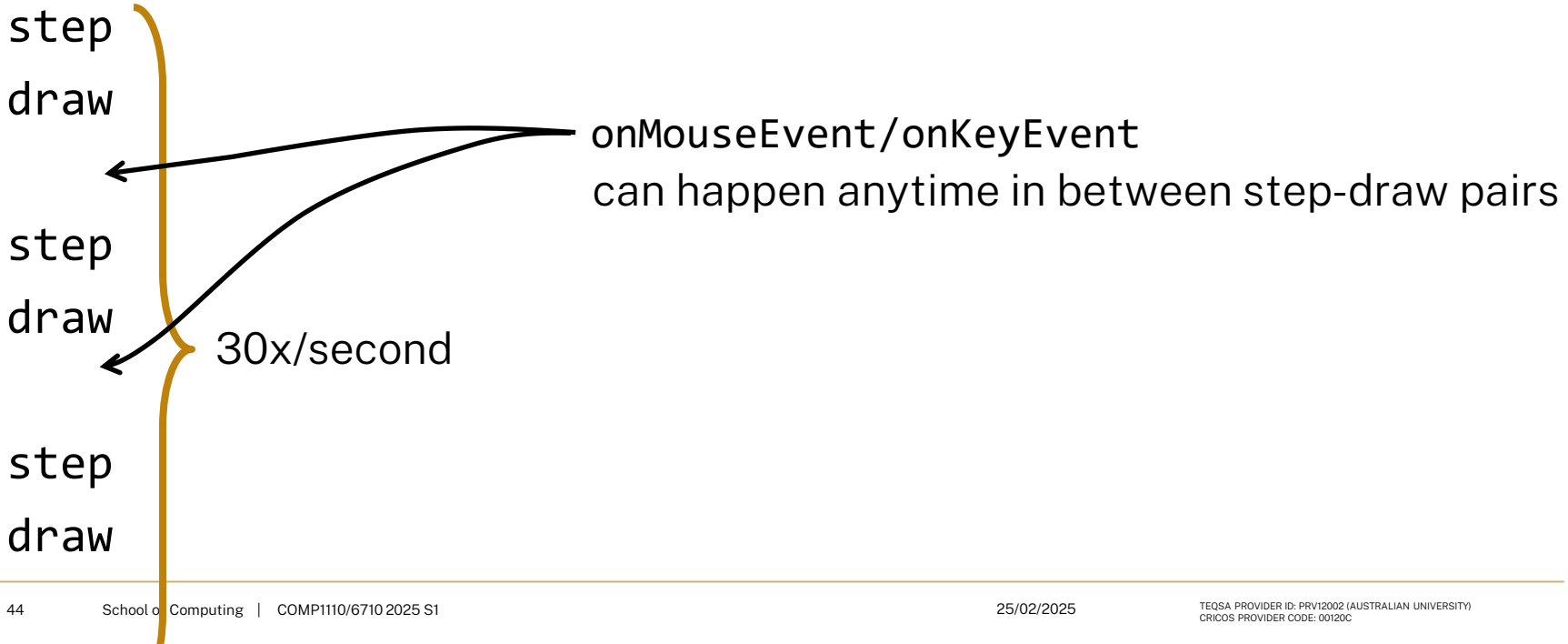
See library documentation on Functional Java Page.

BigBang

```
void main() {
        var initialState = …;
        BigBang(“[title of window]”, initialState, this::draw,
                        this::step, this::onKeyEvent,
                        this::onMouseEvent, this::hasEnded);
}
```


BigBang – Evaluation Model

```
draw(initialState)
```



Practice

Fork and clone the comp1110-2025s1-workshops project.
Create a folder “ws2a”, and work in “World.java” in there.
Commit and push when you are done.

Design a World Program in stages.

Stage 1: the player controls a rectangle that starts in the middle of the world, and can be moved up, down, left, and right with the arrow keys.

Stage 2: on a click, the square turns into a circle. On another click, the circle turns back into a rectangle. The circle can move just like the rectangle.

Stage 3: if there is currently a rectangle, pressing the “w” key increases its width, pressing “e” reduces the width, “h” increases the height, and “j” decreases the height. When changing from a circle to a rectangle, the rectangle’s width and height are reset to their original values.

Stage 4: if there is currently a circle, pressing “r” increases its radius, and pressing “e” decreases the radius. When changing to a circle, the circle’s radius is the smaller of the rectangle’s width and height.