CS2030 Lecture 1

Refresher in Programming

Henry Chia (hchia@comp.nus.edu.sg)

Semester 1 2023 / 2024

Outline and Learning Outcomes

- Refresh on basic programming constructs and the *data–process* model of computational problem solving
- Familiarity with *statically-typed* values and variables
- Instill a sense of *type awareness* when developing programs using a strongly-typed language
- Understand the concept of *abstraction*
  - *functional abstraction* and *data abstraction*
- Understand program execution using the Java memory model
- Appreciate the motivation behind effect-free programming
- Appreciate the difference between program *interpretation* (*translation*) and program *compilation*

Data–Process Model

- Data
  - Primitive: numerical, character, boolean
  - Reference: for composite data
    - Homogeneous: array (multi-dimensional)
    - Heterogeneous: record
- Process
  - Primitive operations: arithmetic, relational, logical, ...
  - Control structures: sequence, selection, repetition
  - Modularity: value-returning function and procedure
  - Input and output
- Coding Style: `https://www.comp.nus.edu.sg/~cs2030/style/`

Interpreter for Java — JShell

- `JShell` (introduced since Java 9) provides an interactive shell
  - uses REPL to provide an immediate feedback loop

```
$ jshell
|  Welcome to JShell -- Version 17
|  For an introduction type: /help intro

jshell> 1 + 1
$1 ==> 2

jshell> /exit
|  Goodbye
```

- Useful for testing language constructs and prototyping
- JShell will be used extensively as a testing framework
  - unit testing
  - incremental (integrated) testing

Assignment with Typed Values and Variables

- Dynamic Typing (e.g. Python):

```
>>> a = 5.0
>>> b = "Hello"
>>> a = b
>>> a
'Hello'
```

- Static Typing (e.g. Java):

```
jshell> double a = 5.0
a ==> 5.0

jshell> String b = "Hello"
b ==> "Hello"

jshell> a = b
|  Error:
|  incompatible types: java.lang.String cannot be converted to double
|  a = b
```

- Java is a type-safe language — strict type checking
- Need to develop a sense of *type awareness*

Abstraction

- Reduce complexity by filtering out unnecessary details
- Exercise: a point comprises of two **double** floating point values representing the x and y coordinates
  - computing the Euclidean distance d between (p_x, p_y) and (q_x, q_y)

$$d = \sqrt{(q_x - p_x)^2 + (q_y - p_y)^2}$$

  - e.g. distance between $p = (0, 0)$ and $q = (1, 1)$

```
jshell> double dx = 1.0 - 0.0
dx ==> 1.0

jshell> double dy = 1.0 - 0.0
dy ==> 1.0

jshell> double distance = Math.sqrt(dx * dx + dy * dy)
distance ==> 1.4142135623730951
```


Functional Abstraction

- *Modularity*: define a *generalized* and *cohesive* task
- A *module/function* (or *method* in Java) can take the form of
  - a *function* that returns *exactly one* value; or

```
jshell> double distanceBetween(double p_x, double p_y,
   ...>    double q_x, double q_y) {
   ...>    double dx = q_x - p_x;
   ...>    double dy = q_y - p_y;
   ...>    return Math.sqrt(dx * dx + dy * dy);
   ...> }
|  created method distanceBetween(Point,Point)

jshell> double distance = distanceBetween(origin, new Point(1.0, 1.0))
distance ==> 1.4142135623730951
```

  - a *procedure* that does something but returns nothing (**void**)

```
jshell> void printHello() {
   ...>    System.out.println("Hello");
   ...> }
|  created method printHello()

jshell> printHello()
Hello
```


Data Abstraction

- Create a `Point` record (since Java 14) as a *user-defined type*

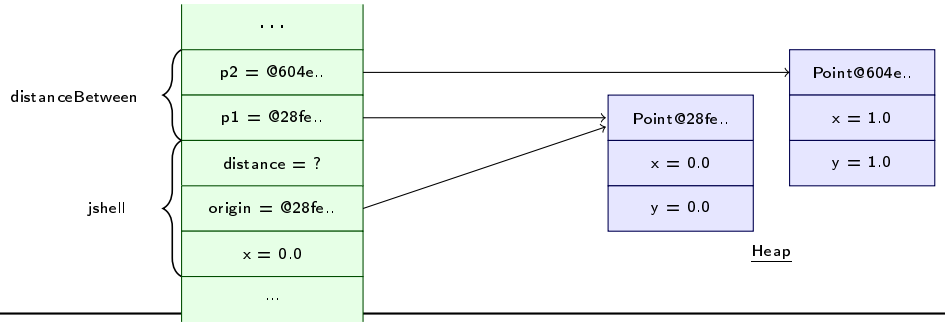
```
jshell> record Point(double x, double y) {}
|  created record Point

jshell> Point origin = new Point(0.0, 0.0)
origin ==> Point[x=0.0, y=0.0]

jshell> origin.x()
$.. ==> 0.0

jshell> double distanceBetween(Point p1, Point p2) {
   ...>    double dx = p2.x() - p1.x();
   ...>    double dy = p2.y() - p1.y();
   ...>    return Math.sqrt(dx * dx + dy * dy);
   ...> }
|  created method distanceBetween(Point,Point)

jshell> double distance = distanceBetween(origin, new Point(1.0, 1.0))
distance ==> 1.4142135623730951
```

- Record classes are a special kind of class
  - model plain data aggregates with *less ceremony than normal classes*

Java Memory Model

- LIFO *stack* for storing activation records of method calls
- *Heap* for storing Java "objects" created via **new**
- E.g. memory model *just before* `distanceBetween` returns

```
jshell> double x = 0.0
x ==> 0.0

jshell> Point origin = new Point(x, 0.0)
origin ==> Point[x=0.0, y=0.0]

jshell> double distance = distanceBetween(origin, new Point(1.0, 1.0)) // pass-by-(address)-value
distance ==> 1.4142135623730951
```



Composite Data: `ArrayList`

- Java's `ArrayList` can be used to represent an *abstraction* of an array
  - need not know how the collection is implemented
- Represent all points using a list of points: `ArrayList<Point>`

```
jshell> ArrayList<Point> points = new ArrayList<Point>()
points ==> []

jshell> points.add(origin)
$.. ==> true

jshell> points.add(new Point(1.0, 1.0))
$.. ==> true

jshell> points
points ==> [Point[x=0.0, y=0.0], Point[x=1.0, y=1.0]]

jshell> points.get(1)
$.. ==> Point[x=1.0, y=1.0]

jshell> points.get(1).x()
$.. ==> 1.0
```

- `ArrayList` is a *generic* type — a container type containing *any specified* type, e.g. `ArrayList<Point>`

Effect-free Programming

- Ensure that all data are **immutable**
  - prevents internal data values being modified once initialized

```
jshell> record Point(double x, double y) {}
|  created record Point

jshell> Point origin = new Point(0.0, 0.0)
origin ==> Point[x=0.0, y=0.0]

jshell> origin.x()
$.. ==> 0.0

jshell> origin.x
|  Error:
|  x has private access in Point
|  origin.x
|  ^------^

jshell> record Point(double x, double y) {
   ...>     void foo() { x = x + 1;}
   ...> }
|  Error:
|  cannot assign a value to final variable x
|      void foo() { x = x + 1;}
|
```

- Facilitates code *maintainability* and *testability*

`ImList`: an Effect-Free List

- `ArrayList`'s `add(..)` has a *side-effect*

```
jshell> points
points ==> [Point[x=0.0, y=0.0], Point[x=1.0, y=1.0]]

jshell> Point p = new Point(2.0, 2.0)
p ==> Point[x=2.0, y=2.0]

jshell> points.add(p)
$.. ==> true

jshell> points.size()
$.. ==> 3

jshell> points // internal state of points is modified!
points ==> [Point[x=0.0, y=0.0], Point[x=1.0, y=1.0], Point[x=2.0, y=2.0]]
```

- Effect-free programming: use immutable list `ImList` instead

```
jshell> ImList<Point> pts = new ImList<Point>(points) // creates an ImList from any Java list
pts ==> [Point[x=0.0, y=0.0], Point[x=1.0, y=1.0]]

jshell> pts.add(p)
$.. ==> [Point[x=0.0, y=0.0], Point[x=1.0, y=1.0], Point[x=2.0, y=2.0]]

jshell> pts
pts ==> [Point[x=0.0, y=0.0], Point[x=1.0, y=1.0]]

jshell> pts = pts.add(p)
pts ==> [Point[x=0.0, y=0.0], Point[x=1.0, y=1.0], Point[x=2.0, y=2.0]]

jshell> pts
pts ==> [Point[x=0.0, y=0.0], Point[x=1.0, y=1.0], Point[x=2.0, y=2.0]]
```

`ImList`: Read/Write-Access

- Write-access: `add`, `remove`, `set`, ... returns a new list

```
jshell> ImList<Integer> list = new ImList<Integer>().add(1).add(2).add(3)
list ==> [1, 2, 3]

jshell> list.add(4)
$.. ==> [1, 2, 3, 4]

jshell> list.remove(1)
$.. ==> [1, 3]

jshell> list.remove(3)
$.. ==> [1, 2, 3]

jshell> list.set(1, 4)
$.. ==> [1, 4, 3]
```

- Read-access: `get`, `size`, `isEmpty`, ... returns a value

```
jshell> list.get(0)
$.. ==> 1

jshell> list.size()
$.. ==> 3

jshell> list.isEmpty()
$.. ==> false

jshell> new ImList<String>().size()
$.. ==> 0

jshell> new ImList<String>().isEmpty()
$.. ==> true
```


Exercise

- Find the maximum distance between any two points from a given list of points, `pts`

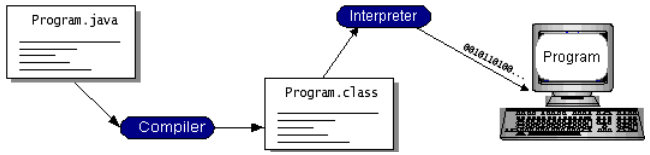
```
jshell> double findMaxDistance(ImList<Point> pts) {
   ...>     double maxDistance = 0.0;
   ...>
   ...>     for (int i = 0; i < pts.size() - 1; i++) {
   ...>        for (int j = i + 1; j < pts.size(); j++) {
   ...>           double distance = distanceBetween(pts.get(i), pts.get(j));
   ...>           if (distance > maxDistance) {
   ...>              maxDistance = distance;
   ...>           }
   ...>        }
   ...>     }
   ...>     return maxDistance;
   ...> }
|  created method findMaxDistance(ImList<Point>)

jshell> pts
pts ==> [Point[x=0.0, y=0.0], Point[x=1.0, y=1.0], Point[x=2.0, y=2.0]]

jshell> double maxDistance = findMaxDistance(points)
maxDistance ==> 2.8284271247461903
```


Interpretation vs Compilation

- While JShell *interprets* Java code snippets, Java programs can also be *compiled* and executed via a driver class

```
record Point(double x, double y) {}

class Program { // driver class with a main method

    static double distanceBetween(Point p1, Point p2) { // note static modifier
        double dx = p2.x() - p1.x();
        double dy = p2.y() - p1.y();
        return Math.sqrt(dx * dx + dy * dy);
    }

    static double findMaxDistance(ImList<Point> pts) { // note static modifier
        double maxDistance = 0.0;

        for (int i = 0; i < pts.size() - 1; i++) {
            for (int j = i + 1; j < pts.size(); j++) {
                double distance = distanceBetween(pts.get(i), pts.get(j));
                if (distance > maxDistance) {
                    maxDistance = distance;
                }
            }
        }
        return maxDistance;
    }

    public static void main(String[] args) { // first method to run
        ImList<Point> pts = new ImList<Point>().add(new Point(0.0, 0.0)).
            add(new Point(1.0, 1.0)).add(new Point(2.0, 2.0));
        double maxDistance = findMaxDistance(pts);
        System.out.println(maxDistance);
    }
}
```


Compiling and Running a Java Program



- To compile (assuming saved in `Program.java`):

```
$ javac Point.java Program.java
```

  - Syntax errors or incompatible typing throws off a compile-time error

- Bytecode created (`Program.class`) translated and executed/run on the Java Virtual Machine (JVM) using:

```
$ java Program
2.8284271247461903
```