

<div data-bbox="56 159 537 287" data-label="Section-Header"> <h1>CS2030 Lecture 1</h1> <h2>Refresher in Programming</h2> </div> <div data-bbox="56 359 616 406" data-label="Text"> <p>Henry Chia (hchia@comp.nus.edu.sg)</p> </div> <div data-bbox="56 462 369 502" data-label="Text"> <p>Semester 2 2022 / 2023</p> </div>	<div data-bbox="1176 15 1635 63" data-label="Section-Header"> <h3>Data–Process Model</h3> </div> <div data-bbox="1176 127 2105 694" data-label="List-Group"> <ul style="list-style-type: none"> <li>□ Data (Memory) <ul style="list-style-type: none"> <li>– Primitive (Singleton): numerical, character, boolean</li> <li>– Reference: for composite data <ul style="list-style-type: none"> <li>▸ Homogeneous: array (multi-dimensional)</li> <li>▸ Heterogeneous: record</li> </ul> </li> </ul> </li> <li>□ Process (Mechanism) <ul style="list-style-type: none"> <li>– Primitive operations: arithmetic, relational, logical, ...</li> <li>– Control structures: sequence, selection, repetition</li> <li>– Modularity: value-returning function and procedure</li> <li>– Input and output</li> </ul> </li> <li>□ Coding Style: <a href="https://www.comp.nus.edu.sg/~cs2030/style/">https://www.comp.nus.edu.sg/~cs2030/style/</a></li> </ul> </div>
<div data-bbox="1019 742 1086 774" data-label="Page-Footer"> <p>1 / 16</p> </div>	<div data-bbox="2139 742 2206 774" data-label="Page-Footer"> <p>3 / 16</p> </div>
<div data-bbox="56 813 761 869" data-label="Section-Header"> <h3>Outline and Learning Outcomes</h3> </div> <div data-bbox="56 925 1052 1452" data-label="List-Group"> <ul style="list-style-type: none"> <li>□ Refresh on basic programming constructs and the <i>data–process</i> model of computational problem solving</li> <li>□ Familiarity with <i>statically-typed</i> values and variables</li> <li>□ Instill a sense of <i>type awareness</i> when developing programs using a strongly-typed language</li> <li>□ Understand the concept of <i>abstraction</i> <ul style="list-style-type: none"> <li>– <i>functional abstraction</i> and <i>data abstraction</i></li> </ul> </li> <li>□ Understand program execution using the Java memory model</li> <li>□ Appreciate the motivation behind effect-free programming</li> <li>□ Appreciate the difference between program <i>interpretation</i> (<i>translation</i>) and program <i>compilation</i></li> </ul> </div>	<div data-bbox="1176 813 1825 869" data-label="Section-Header"> <h3>Interpreter for Java — JShell</h3> </div> <div data-bbox="1176 925 2184 1508" data-label="List-Group"> <ul style="list-style-type: none"> <li>□ JShell (introduced since Java 9) provides an interactive shell <ul style="list-style-type: none"> <li>– uses REPL to provide an immediate feedback loop <pre> \$ jshell   Welcome to JShell -- Version 17   For an introduction type: /help intro  jshell&gt; 1 + 1 \$1 ==&gt; 2  jshell&gt; /exit   Goodbye </pre> </li> </ul> </li> <li>□ Useful for testing language constructs and prototyping</li> <li>□ JShell will be used extensively as a testing framework <ul style="list-style-type: none"> <li>– unit testing</li> <li>– incremental (integrated) testing</li> </ul> </li> </ul> </div>
<div data-bbox="1019 1540 1086 1572" data-label="Page-Footer"> <p>2 / 16</p> </div>	<div data-bbox="2139 1540 2206 1572" data-label="Page-Footer"> <p>4 / 16</p> </div>

# 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*

5 / 16

# 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
```

7 / 16

# 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
```

6 / 16

# Data Abstraction

- Create a **Point** class as a *user-defined type*

```
jshell> class Point {
...> double x; // fields/properties
...> double y;
...>
...> Point(double x, double y) { // constructor
...> this.x = x;
...> this.y = y;
...> }
...> }
| created class Point

jshell> Point origin = new Point(0.0, 0.0) // creating a point
origin ==> Point@28feb3fa

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
```

8 / 16

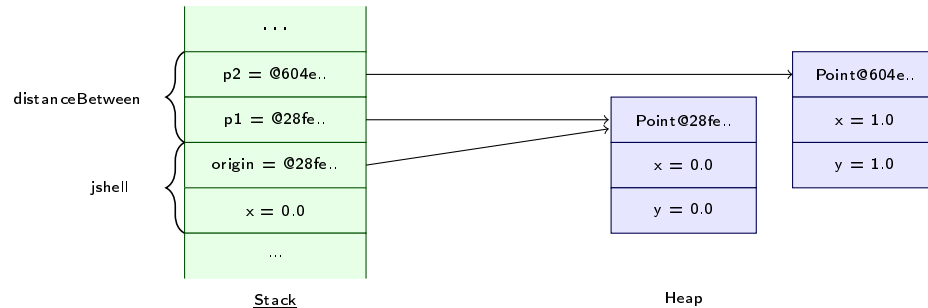
# Java Memory Model

- 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@28feb3fa

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



9 / 16

# Effect-free Programming

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

```
jshell> points
points ==> [Point@28feb3fa, Point@675d3402]

jshell> Point p = new Point(2.0, 2.0)
p ==> Point@51565ec2

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

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

jshell> points // internal state of points is modified!
points ==> [Point@28feb3fa, Point@675d3402, Point@51565ec2]
```

- 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@28feb3fa, Point@675d3402]

jshell> pts.add(p) // add(p) returns a new list
?.. ==> [Point@28feb3fa, Point@675d3402, Point@51565ec2]

jshell> pts // internal state unchanged, hence immutable
pts ==> [Point@28feb3fa, Point@675d3402]

jshell> pts.size()
?.. ==> 2

jshell> pts = pts.add(p) // pts re-assigned to a new list
pts ==> [Point@28feb3fa, Point@675d3402, Point@51565ec2]
```

11 / 16

## 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@28feb3fa, Point@675d3402]

jshell> points.get(1)
$.. ==> Point@675d3402

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

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

10 / 16

## Our Own Immutable `ImList`

- 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
```

12 / 16

## Immutability in a User-Defined Type

- To make `Point` immutable, declare properties as **final**

```
class Point {  
    final double x;  
    final double y;  
    ...  
}
```
- Prevents the property values being modified once initialized

```
jshell> Point origin = new Point(0.0, 0.0)  
origin ==> Point@28feb3fa  
  
jshell> origin.x = 2.0  
| Error:  
| cannot assign a value to final variable x  
| origin.x = 2.0  
| ^-----^  
  
jshell> void foo(Point p) {  
| ...> p.x = 2.0;  
| ...> }  
| Error:  
| cannot assign a value to final variable x  
| p.x = 2.0;
```

  - `p` cannot be passed into `foo` for modifications
- Facilitates code *maintainability* and *testability*

13 / 16

## Interpretation vs Compilation

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

```
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);  
    }  
}
```

15 / 16

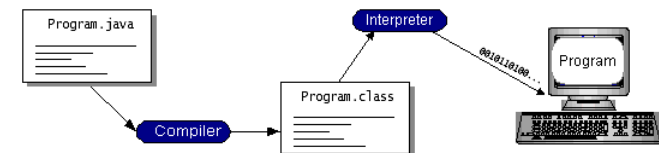
## Exercise

- 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@28feb3fa, Point@675d3402, Point@51565ec2]  
  
jshell> double maxDistance = findMaxDistance(points)  
maxDistance ==> 2.8284271247461903
```

14 / 16

## Compiling and Running a Java Program



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

```
$ javac 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
```

16 / 16