Structured Programming

COMP1110/6710

Australian National University

Image Courtesy NASA/JPL-Caltech.

Object-Oriented

Class-Based

Imperative

Platform-Independent

Industrial

Compiled

Statically-Typed

Garbage-Collected

Classes & Objects

Australian
National
University

Objects

State ✚ Behavior

Objects have identity

- They live on the heap
- They are usually created with **`new`**
- == compares objects for identity

Objects know themselves:

- Their data (state)
- Their operations (behavior)

➔ Different objects may differ both in data and in operations

Classes

Data Description

Subtyping/Polymorphism

Module System

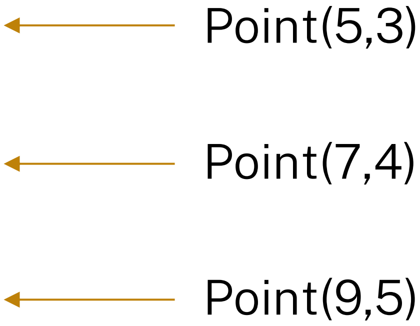
Compilation Unit

Image Author: Picanox; Public Domain

Classes as Data Description

Collections of Objects with the same kinds of data and operations

```
class Point {
    int x;
    int y;
    double getAngle() {
        …
    }
}
```



Point(5,3)

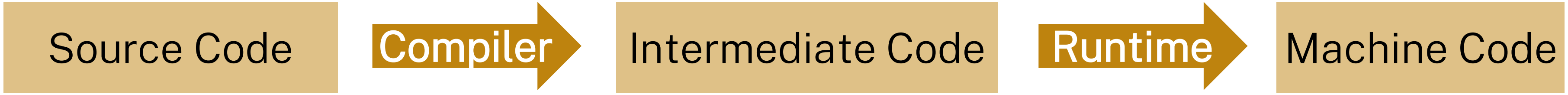
Point(7,4)

Point(9,5)

Classes as Compilation Unit

So far, we always ran a single Java file, but what if you want more?

Java is a compiled language, meaning that creating programs has multiple steps. In Java:



Platform-Independent

Specialized to your
Hardware/Operating System

	As used so far	Java	
javac	--enable-preview –source 23 [yourfile].java	[yourfile(s)].java	Creates class files
java	--enable-preview [yourfile.java]	[class name]	Runs programs

Fields & Constructors

Australian
National
University

Fields

Like Records, Classes have Fields

```
class Point {
    int x;
    int y;
    …
}
```

```
Point p = …; //coming up
int x = p.x; //field access
p.x = x + 3; //assignment
```

Default Constructors

Automatically there if you define no others

```
class Point {
    int x;
    int y;
}
```

```
Point p = new Point();
int x = p.x; //x is 0 here
p.x = x + 3;
p.y = 15;
```

Default Values:

Number types: 0

Booleans: **false**

All others: **null**

Constructors

Initializing your Objects better

```
class Point {
    int x;
    int y;
    Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

```
Point p = new Point(42, 15);
int x = p.x; //x is 42 here
p.x = x + 3;
```

Constructors

You can have several of them

```
class Point {
    int x;
    int y;
    Point(int x, int y) {
       this.x = x;
       this.y = y;
    }
    Point(int c) {
       this.x = c;
       this.y = c;
    }
}
```

```
Point p = new Point(42, 15);
int x = p.x; //x is 42 here
p.x = x + 3;
Point p2 = new Point(11);
p.y = p2.y; //assigns 11
```

Arrays & null

null – the “Billion Dollar Mistake”

Every reference (i.e. non-primitive) type contains one special value: null

Represent uninitialized variables

```
String str; //Declaration Stmt
// str == null at this point
// str.length() causes
// Null Pointer Exception
str = “hello”;
str.length(); // 5
```

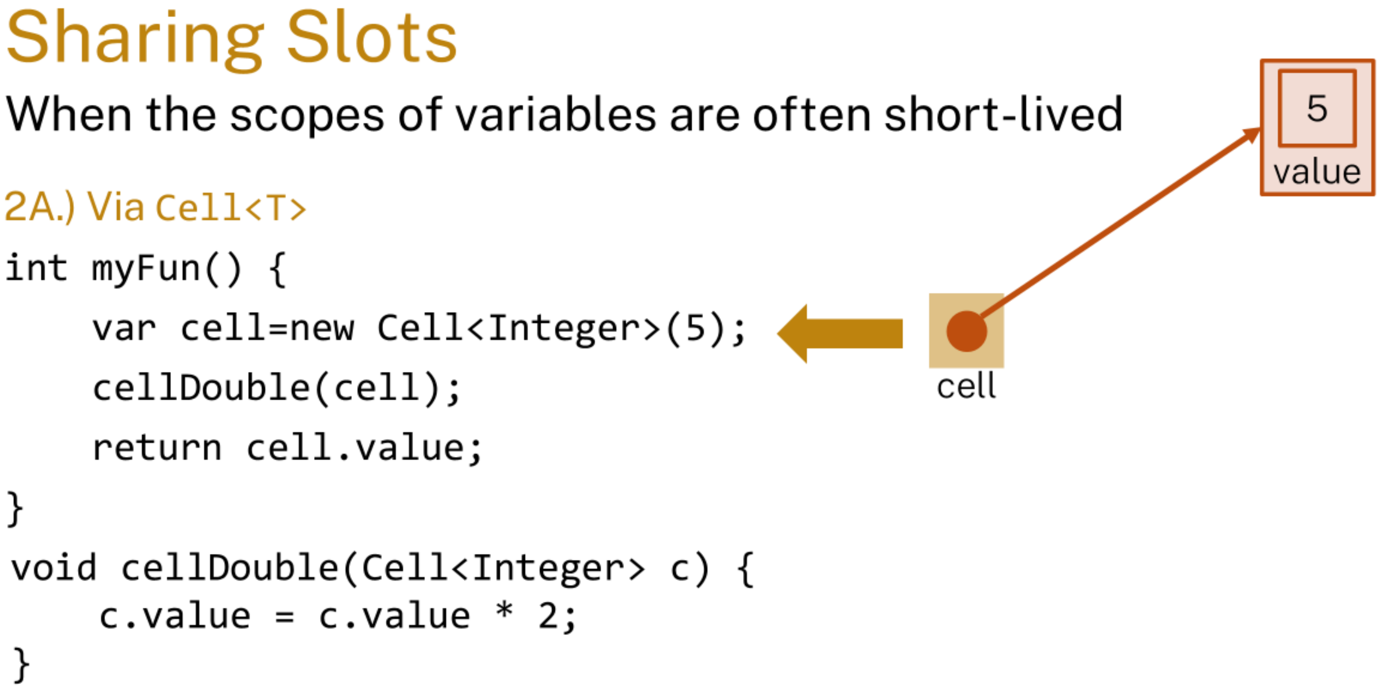
Represent the absence of a value

```
Map<String, String> map =
    new HashMap<>();
String str = map.get(“Hello”);
// str == null at this point
```

Like the Maybe type we’ve seen

Arrays

Storing many values at once



Sharing Slots

When the scopes of variables are often short-lived

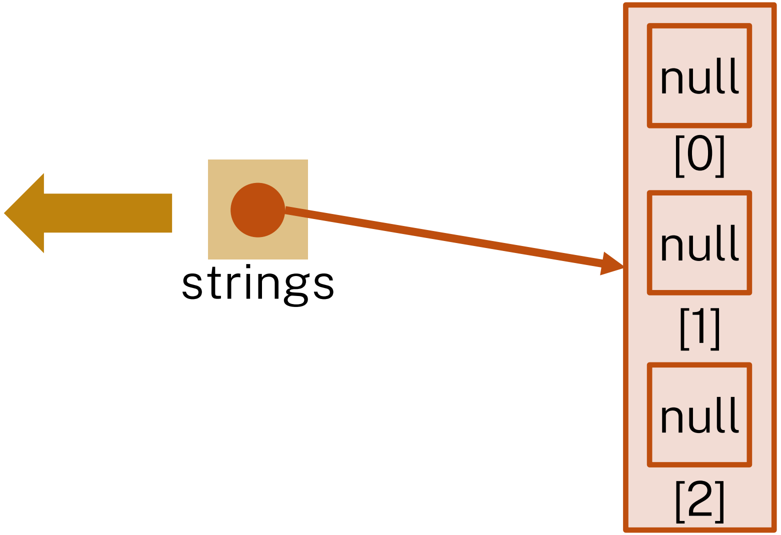
2A.) Via Cell<T>

```
int myFun() {
    var cell=new Cell<Integer>(5);
    cellDouble(cell);
    return cell.value;
}
void cellDouble(Cell<Integer> c) {
    c.value = c.value * 2;
}
```

Arrays

Storing many values at once

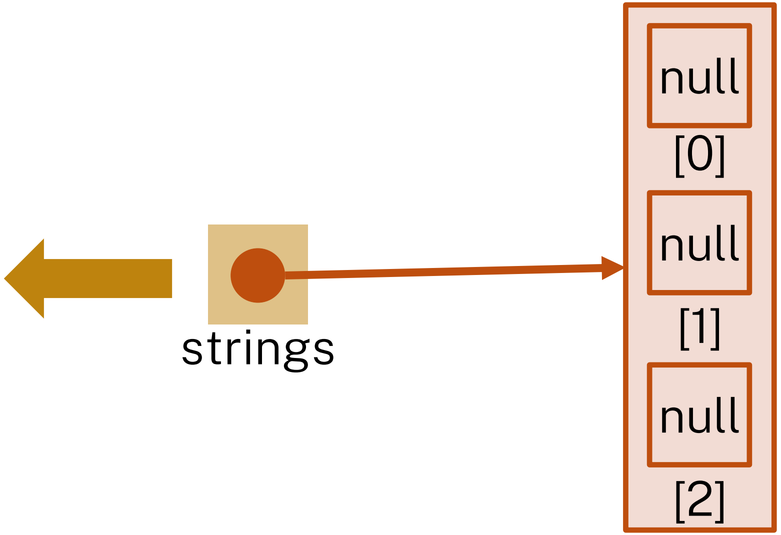
```
String[] strings = new String[3];
strings[0] = “Hello”;
strings[1] = “World”;
strings[2] = “!”;
```



Arrays

Storing many values at once

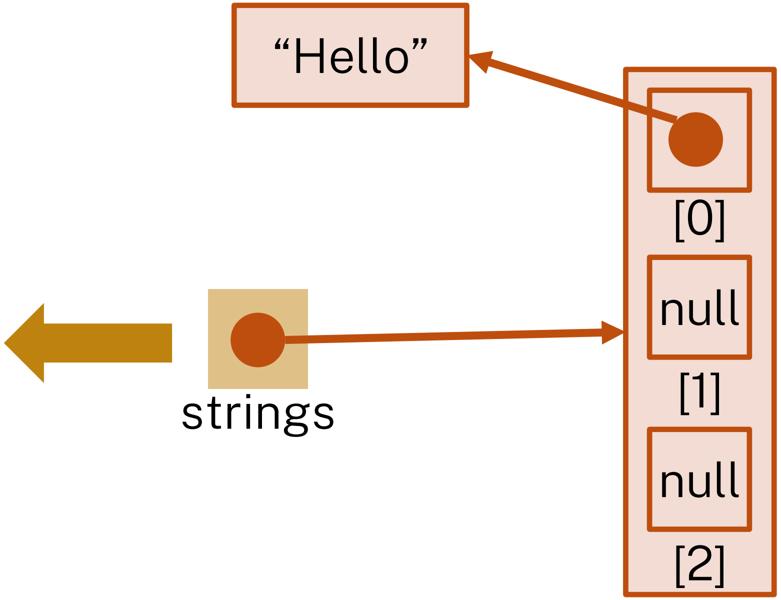
```
String[] strings = new String[3];
strings[0] = “Hello”;
strings[1] = “World”;
strings[2] = “!”;
```



Arrays

Storing many values at once

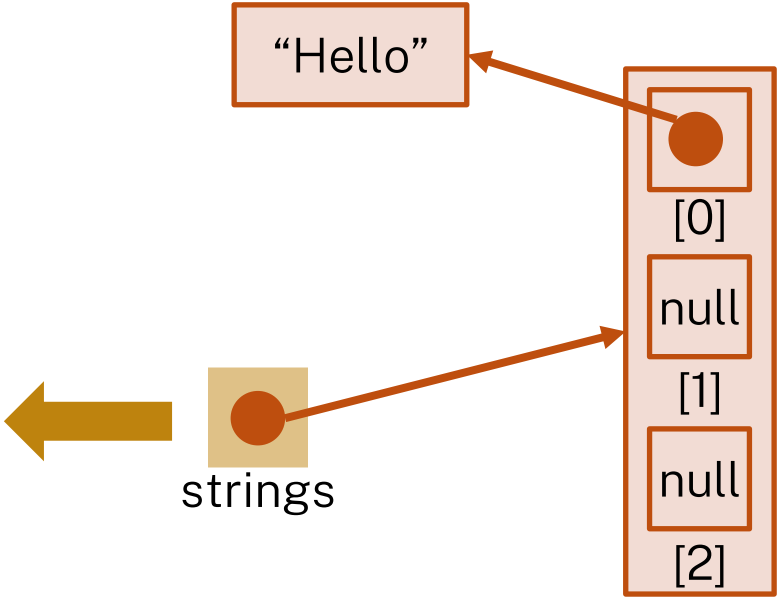
```
String[] strings = new String[3];
strings[0] = “Hello”;
strings[1] = “World”;
strings[2] = “!”;
```



Arrays

Storing many values at once

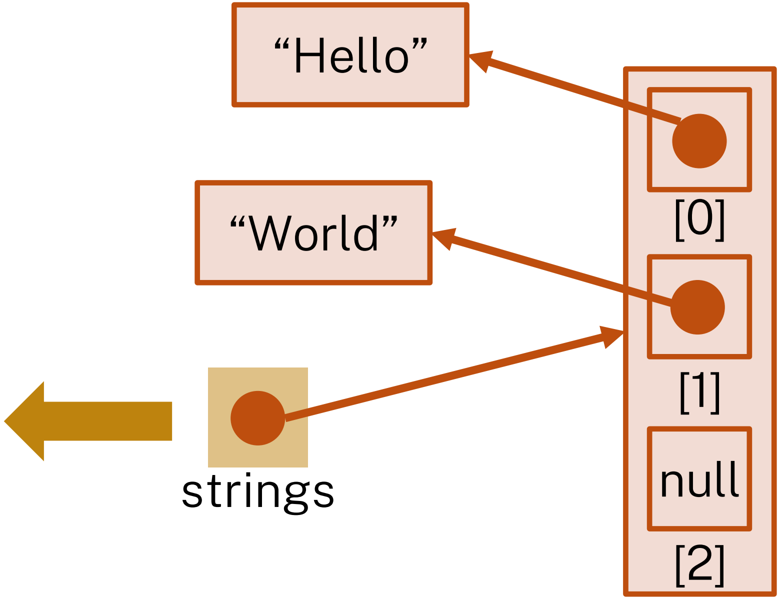
```
String[] strings = new String[3];
strings[0] = “Hello”;
strings[1] = “World”;
strings[2] = “!”;
```



Arrays

Storing many values at once

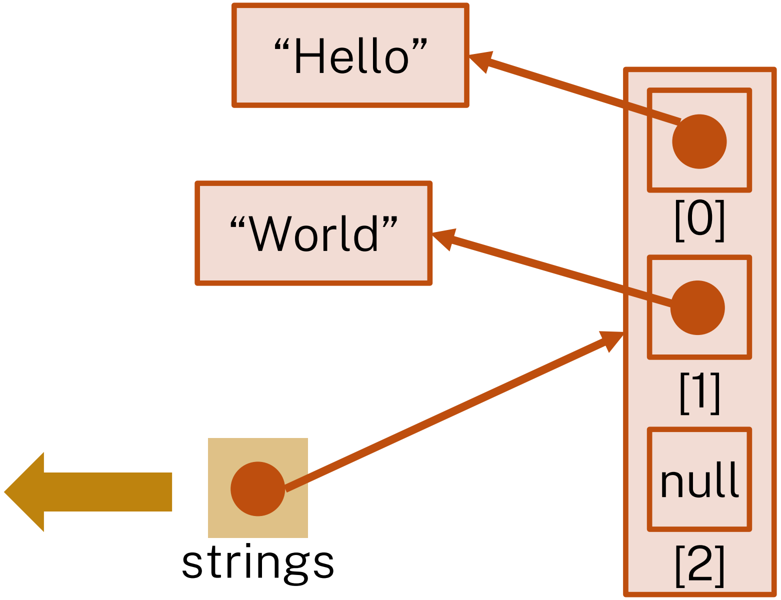
```
String[] strings = new String[3];
strings[0] = “Hello”;
strings[1] = “World”;
strings[2] = “!”;
```



Arrays

Storing many values at once

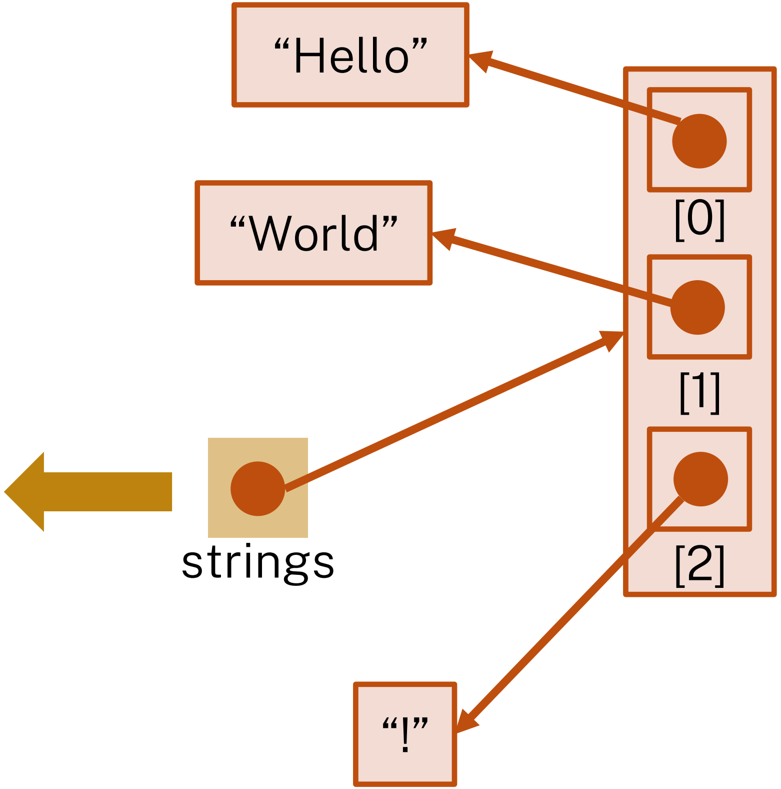
```
String[] strings = new String[3];
strings[0] = “Hello”;
strings[1] = “World”;
strings[2] = “!”;
```



Arrays

Storing many values at once

```
String[] strings = new String[3];
strings[0] = “Hello”;
strings[1] = “World”;
strings[2] = “!”;
```



Alternatively:

```
String[] strings = new
 String[]{“Hello”, “World”, “!”};
```

Loops

An alternative to recursion

For-loops

Where termination is more likely

General for:

```
for(int i=0; i < 10; i++) {
   println(i);
}
```

“Enhanced” for:

```
var strings = MakeList(“A”, B”);
for(var str : strings) {
    println(str);
}
```

Works for many different list-like data types (we’ll see more examples).

For-loops with Arrays

Where termination is more likely

General for:

```
String[] strs = …;
for(int i=0; i < strs.length; i++)
{
   println(strs[i]);
}
```

“Enhanced” for:

```
String[] strings = …;
for(var str : strings) {
    println(str);
}
```

While loops

More general, but much more likely to never stop

while:

```
String[] strs = …;
for(int i=0; i < strs.length; i++)
{
   println(strs[i]);
}
```

do-while:

```
String[] strings = …;
for(var str : strings) {
    println(str);
}
```

Evaluation Order

Not just top-down anymore

Every loop has a condition:

- for: the middle expression e.g. in `for(int i=0; i < 10; i++)`
- enhanced `for`: there are more elements in the list/array
- while/do-while: the boolean expression after `while`

When the condition is true, the loop jumps back to the start

Two special statements:

`break;` - end the loop right now, ignoring the condition

`continue;` - go back to the start of the loop right now – in for-loops, use next element

Methods

Functions Associated With a Data Type

Methods

Functions Associated With a Data Type

```
public class Person {
    String firstName;
    String lastName;
    String getFullName() {
        return firstName + this.lastName;
    }
}
```

Objects know themselves. Can be omitted in many cases (see firstName).

Public Static Void Main

Or: main methods in Java

```
[import statements go here]

public class MyClass {
    public static void main(String[] args) {
        System.out.println(“Hello World!”);
    }
}z
```

In a file called “MyClass.java”

Need to write this in front of every println

Practice

In standard Java:

- Write the hello-world program

In extended functional Java (i.e. with –enable-preview)

- Write a number-guessing game: generate a random number between 1 and 100, and let the user guess numbers, telling them whether the actual number is larger or smaller than their guess, until they got the right answer.
- Write a tree class (with arbitrary numbers of children per node) that has a method which returns how many nodes it has, i.e. `int countNodes()` [no arguments]