Needs ANU Account!

Australian National University

Structured Programming

COMP1110/6710

pollev.com/fabianm
Register for Engagement

Image Courtesy NASA/JPL-Caltech.

Recap: Assignments

Three steps you must do:

- Push Code on GitLab (can use assignment variables/extensions)
- Register for Code Walk (Base Deadline Day, 18:00, no extensions)
- Attend Code Walk at Scheduled Time

Two optional steps:

- Provide Particular Commit You Want to Submit (otherwise, latest commit before base deadline)
- Provide Scheduling Preferences for Code Walk (otherwise, some time during your registered tutorial time)

TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

Mid-Term Test

Next Monday, 24/3 COMP1110: 14:00 COMP6710: 17:30

No Materials! We'll give you the Functional Java Standard Library Documentation, as well as the Java Standard Library Documentation from Oracle.

You can use Functional Java, regular Java, or any mix.

Make sure to produce working code!

TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

Mid-Term Test

Next Monday, 24/3 COMP1110: 14:00 COMP6710: 17:30

No Materials! Except Dictionaries with School Approval.

If you have a permission slip to use a dictionary, drop off your dictionary at the Exams Office (Melville Hall) by Thursday (this is a change from what was previously announced, because we got central invigilation now).

TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

(Mutable) State (continued)

The stuff that some of you have already been using, even though they should not have

Still illegal in P2, and only partially allowed (i.e. using HashMaps) in the last part of U2

Not core material for mid-semester test, but may show up in distinction-level part.

Australian National University

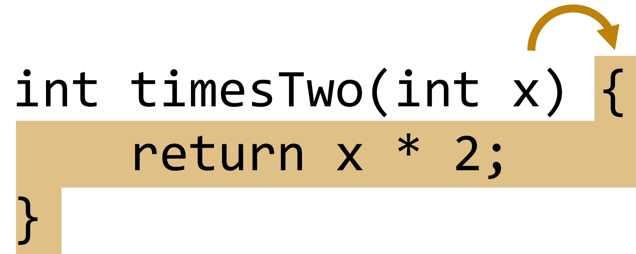
Variables

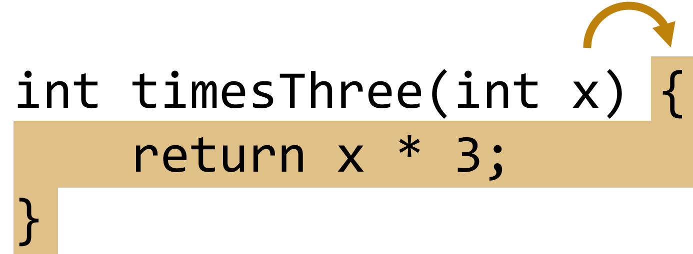
4x2

TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

Variables - Scope

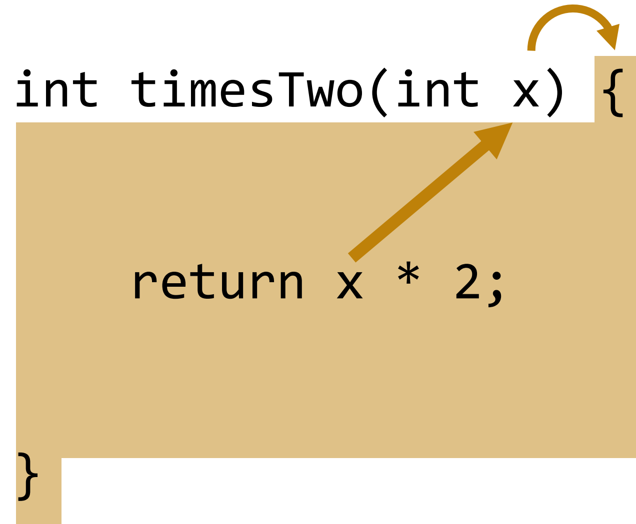
where the same name refers to the same variable

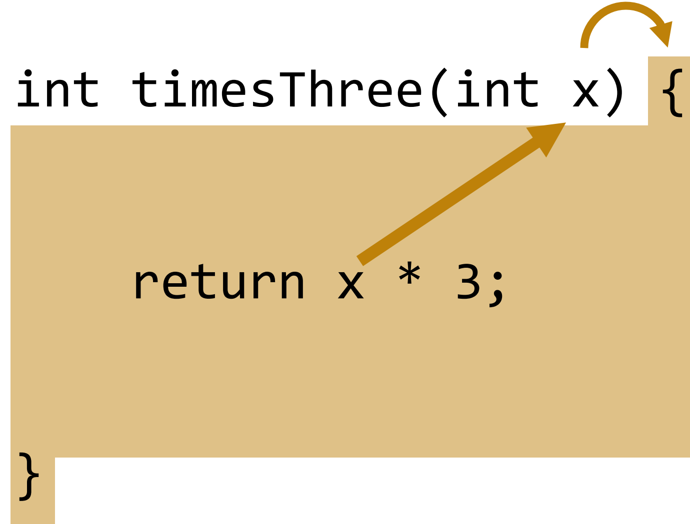




Variables - Scope

where the same name refers to the same variable





Variables - Scope

where the same name refers to the same variable

```
int timesThree(int x) {
    int y = x * 2;
    return y + x;
}
```

Scope of x

Variables - Scope

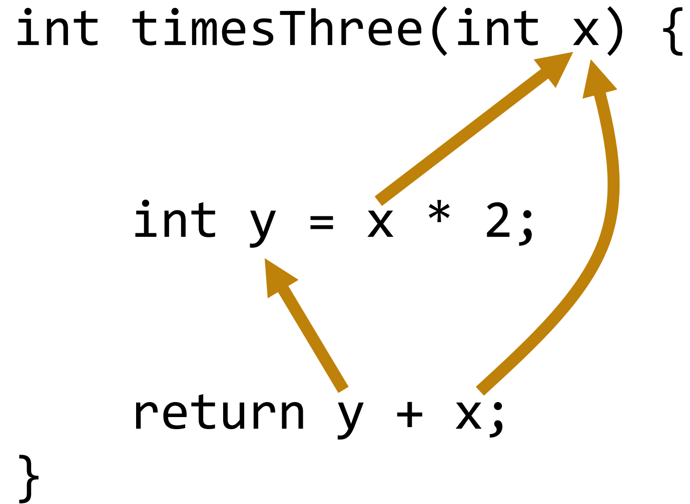
where the same name refers to the same variable

```
int timesThree(int x) {
    int y = x * 2;
    return y + x;
}
```

Scope of y

Variables - Scope

where the same name refers to the same variable



Variables - Scope

where the same name refers to the same variable

```
int someFunction(int x) {
    …
    if(x == 5) {
        int y = 3;
        return x + y;
    } else {
        int y = 4;
        return x + y;
    }
}
```

Scope of x

Variables - Scope

where the same name refers to the same variable

```
int someFunction(int x) {
    …
    if(x == 5) {
        int y = 3;
        return x + y;
    } else {
        int y = 4;
        return x + y;
    }
}
```

y (1)

Scope of y (1)

y (2)

Scope of y (2)

Functional Programming

Each variable represents a **single** value throughout its scope

Imperative Programming

Each variable represents a single **slot** whose content can be accessed and changed throughout its scope

New statement: assignment

`[varName] = [expression]`

e.g. `x = 5 + 2;`

TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

Variables as Slots

someFunction(6) called

```
int someFunction(int x) {
    int y = x + 1;
    x = y + 2;
    y = x - 3;
    x = y * 2;
    y = x - (y + 1);
    return x + y;
}
```

6

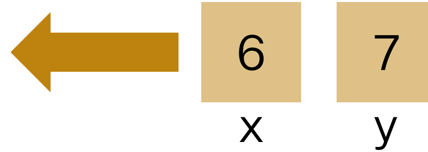
x

TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

Variables as Slots

someFunction(6) called

```
int someFunction(int x) {
    int y = x + 1;
    x = y + 2;
    y = x - 3;
    x = y * 2;
    y = x - (y + 1);
    return x + y;
}
```

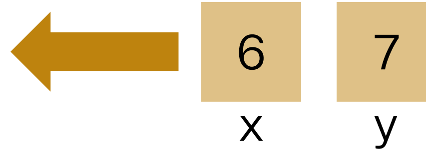


TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

Variables as Slots

someFunction(6) called

```
int someFunction(int x) {
    int y = x + 1;
    x = y + 2;
    y = x - 3;
    x = y * 2;
    y = x - (y + 1);
    return x + y;
}
```

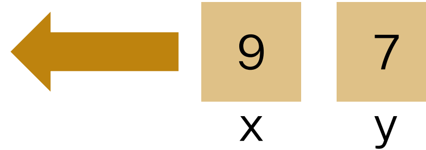


TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

Variables as Slots

someFunction(6) called

```
int someFunction(int x) {
    int y = x + 1;
    x = y + 2;
    y = x - 3;
    x = y * 2;
    y = x - (y + 1);
    return x + y;
}
```

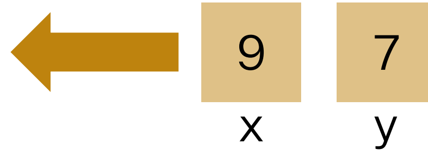


TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

Variables as Slots

someFunction(6) called

```
int someFunction(int x) {
    int y = x + 1;
    x = y + 2;
    y = x - 3;
    x = y * 2;
    y = x - (y + 1);
    return x + y;
}
```

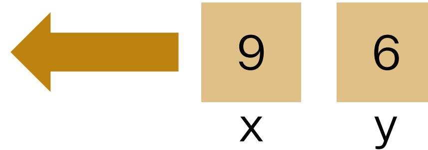


TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

Variables as Slots

someFunction(6) called

```
int someFunction(int x) {
    int y = x + 1;
    x = y + 2;
    y = x - 3;
    x = y * 2;
    y = x - (y + 1);
    return x + y;
}
```

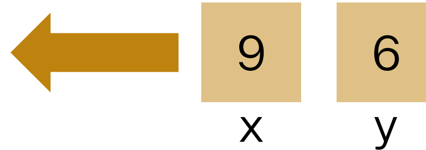


TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

Variables as Slots

someFunction(6) called

```
int someFunction(int x) {
    int y = x + 1;
    x = y + 2;
    y = x - 3;
    x = y * 2;
    y = x - (y + 1);
    return x + y;
}
```

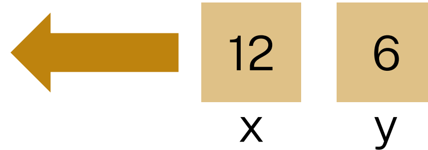


TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

Variables as Slots

someFunction(6) called

```
int someFunction(int x) {
    int y = x + 1;
    x = y + 2;
    y = x - 3;
    x = y * 2;
    y = x - (y + 1);
    return x + y;
}
```

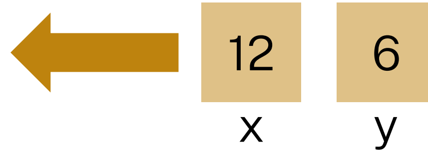


TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

Variables as Slots

someFunction(6) called

```
int someFunction(int x) {
    int y = x + 1;
    x = y + 2;
    y = x - 3;
    x = y * 2;
    y = x - (y + 1);
    return x + y;
}
```

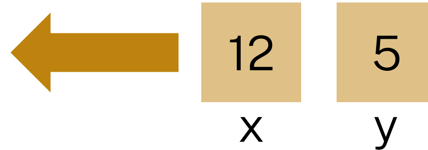


TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

Variables as Slots

someFunction(6) called

```
int someFunction(int x) {
    int y = x + 1;
    x = y + 2;
    y = x - 3;
    x = y * 2;
    y = x - (y + 1);
    return x + y;
}
```

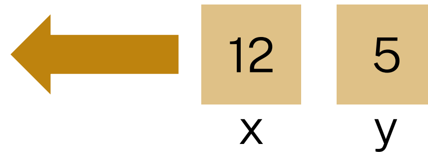


TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

Variables as Slots

someFunction(6) called

```
int someFunction(int x) {
    int y = x + 1;
    x = y + 2;
    y = x - 3;
    x = y * 2;
    y = x - (y + 1);
    return x + y;
}
```

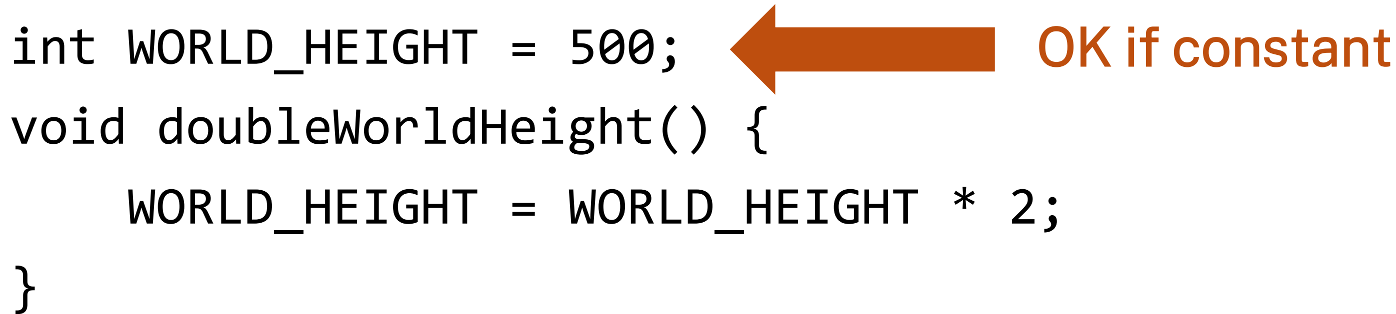


TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

Sharing Slots

When the scopes of variables are often short-lived

1.) Global Variables (mostly a BIIIIIG NO-NO!)



Doing this often causes big issues for maintaining and extending your code. EXTREMELY RARELY a good choice!

TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

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;
}
```

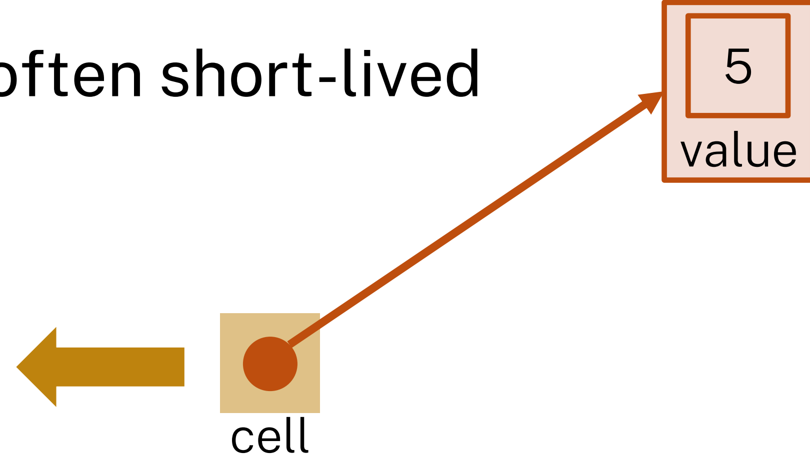

TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

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;
}
```



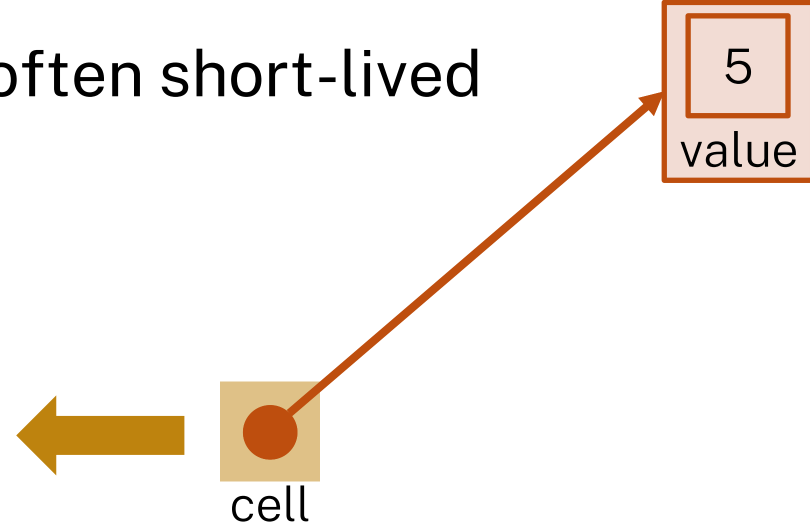
TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

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;
}
```



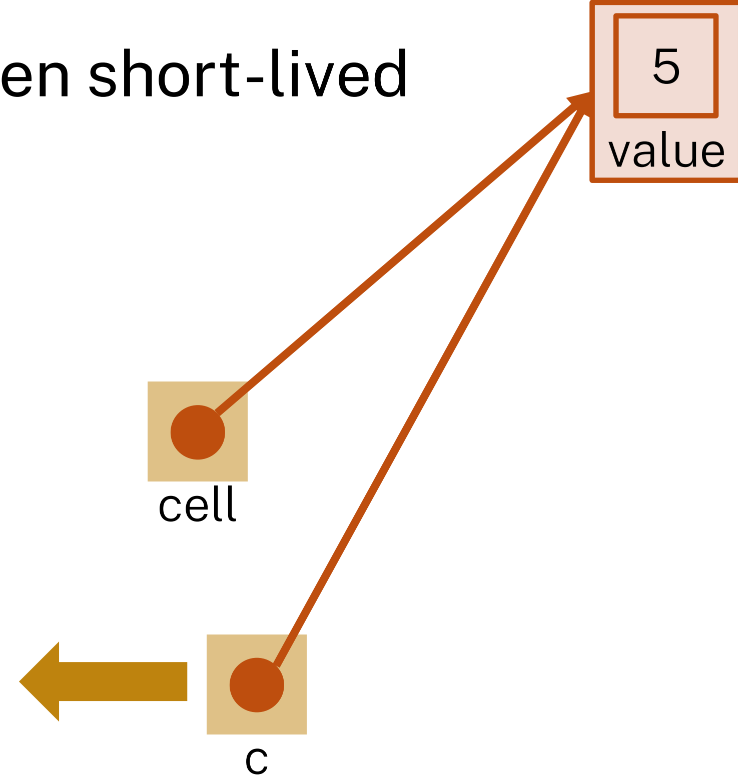
TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

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;
}
```



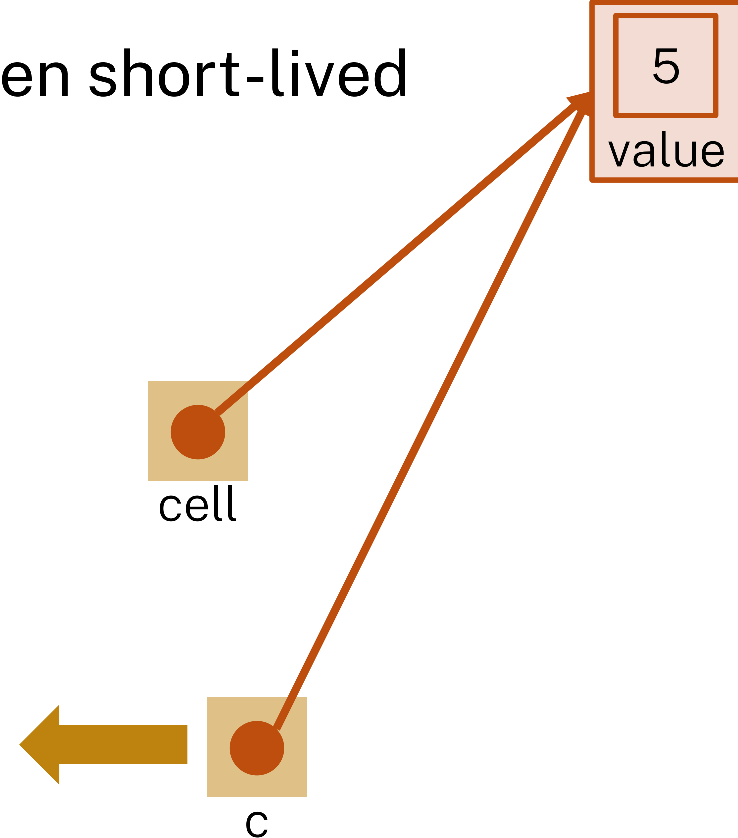
TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

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;
}
```



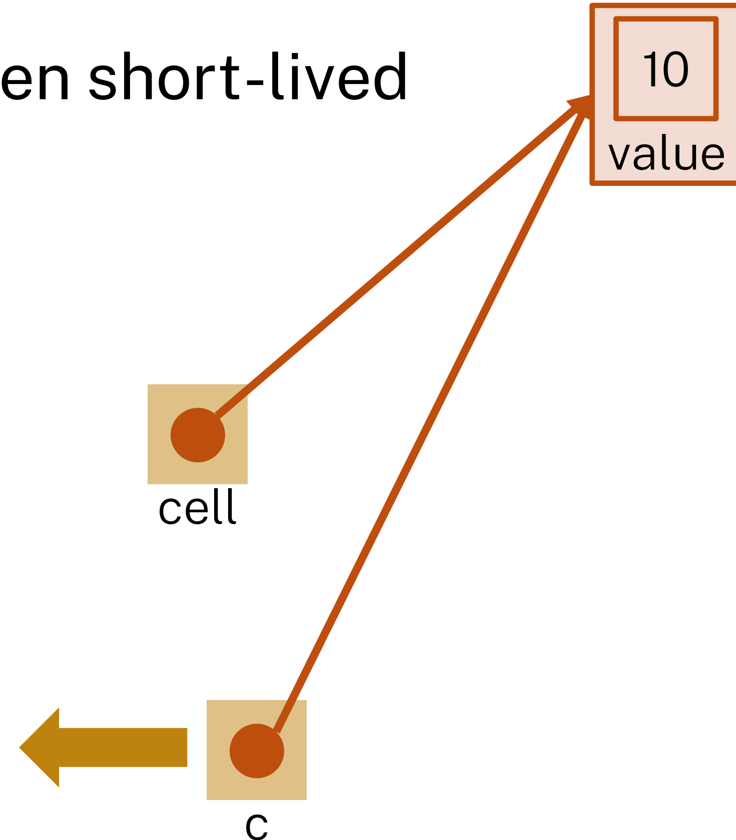
TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

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;
}
```



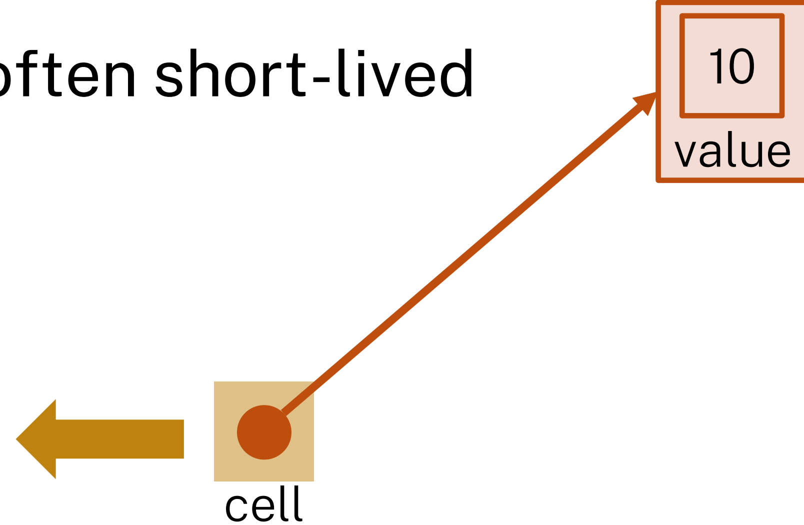
TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

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;
}
```



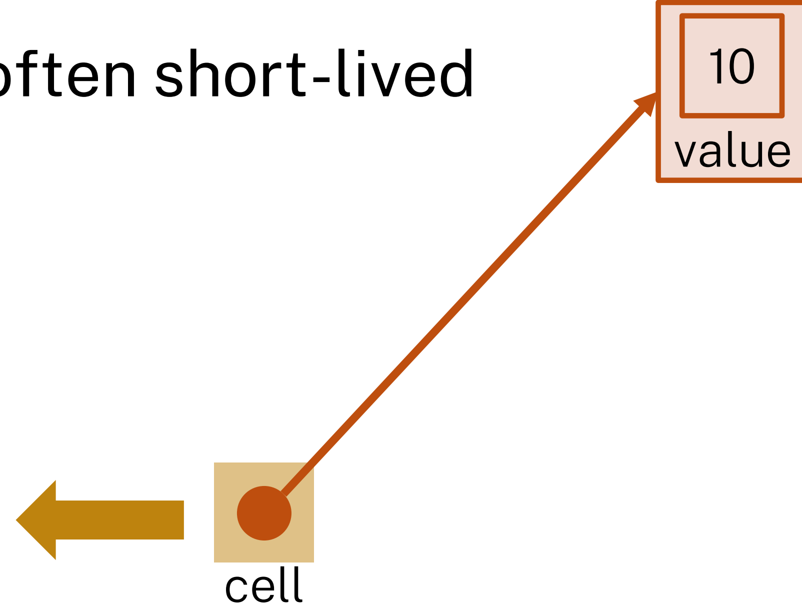
TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

Sharing Slots

When the scopes of variables are often short-lived

2A.) Via `Cell<T>`

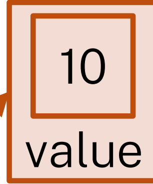
```
int myFun() {
    var cell=new Cell<Integer>(5);
    cellDouble(cell);
    return cell.value;
}
void cellDouble(Cell<Integer> c) {
    c.value = c.value * 2;
}
```



TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

Sharing Slots

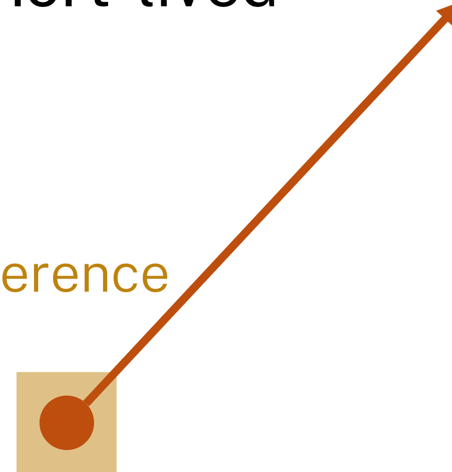
When the scopes of variables are often short-lived

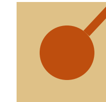


2B.) Via many reference types

Live on the "heap". They exist as long as there are references to them

Live on the "stack". They exist for as long as a variable is in scope





Each can either contain a reference, or a value of one of the primitive value types (lowercase): boolean, byte, char, short, int, long, float, double

Their uppercase versions (Integer, Boolean, etc.) and String are heap values.

TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

Reference Values

The things that live on the heap

Generally created with the “new” keyword.

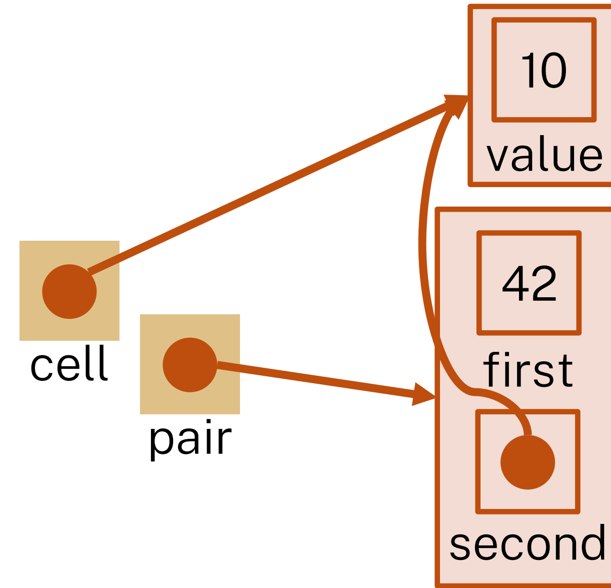
```
var cell = new Cell<>(10);
var pair = new Pair<>(42, cell);
```

But also via functions that internally use “new”:

```
var map = MakeHashMap();
```

And via some special language features:

```
var helloworld = “Hello” + “World”;
Integer myInt = 5;
Function<Integer,Integer> id = x -> x;
```



TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

Functional Programming

ConsList-based Map (CLM)

```
CLM MakeConsMap(Pair<K, V>…)
CLM Put(CLM, K, V)
CLM Remove(CLM, K)
Maybe<V> Get(CLM, K)
boolean ContainsKey(CLM, K)
ConsList<K> GetKeys(CLM)
```

Imperative Programming

HashMap (HM)

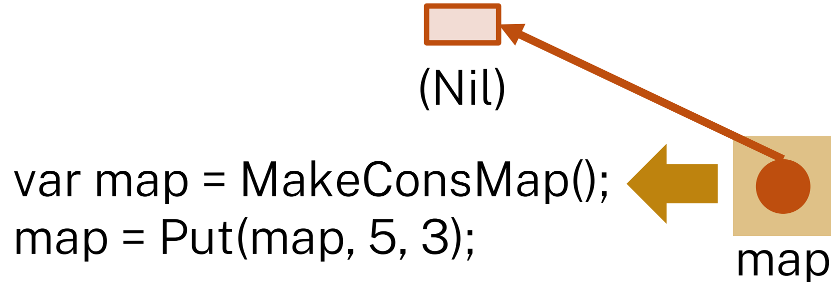
```
HM MakeHashMap(Pair<K, V>…)
void Put(HM, K, V)
void Remove(HM, K)
Maybe<V> Get(HM, K)
boolean ContainsKey(HM, K)
ConsList<K> GetKeys(HM)
```


TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

Functional Programming

ConsList-based Map (CLM)

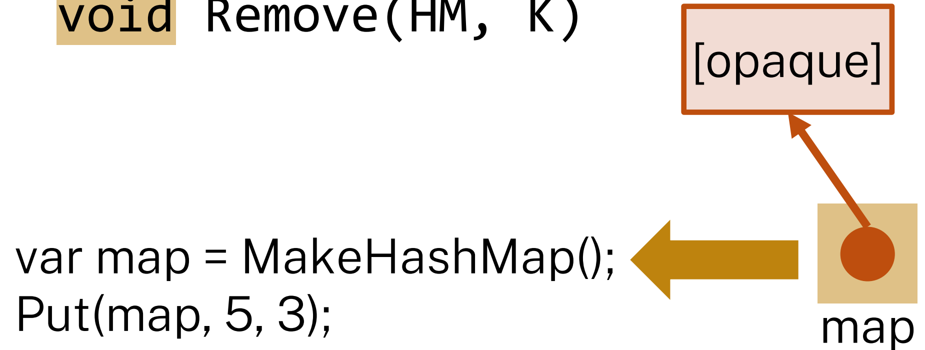
```
CLM Put(CLM, K, V)
CLM Remove(CLM, K)
```



Imperative Programming

HashMap (HM)

```
void Put(HM, K, V)
void Remove(HM, K)
```

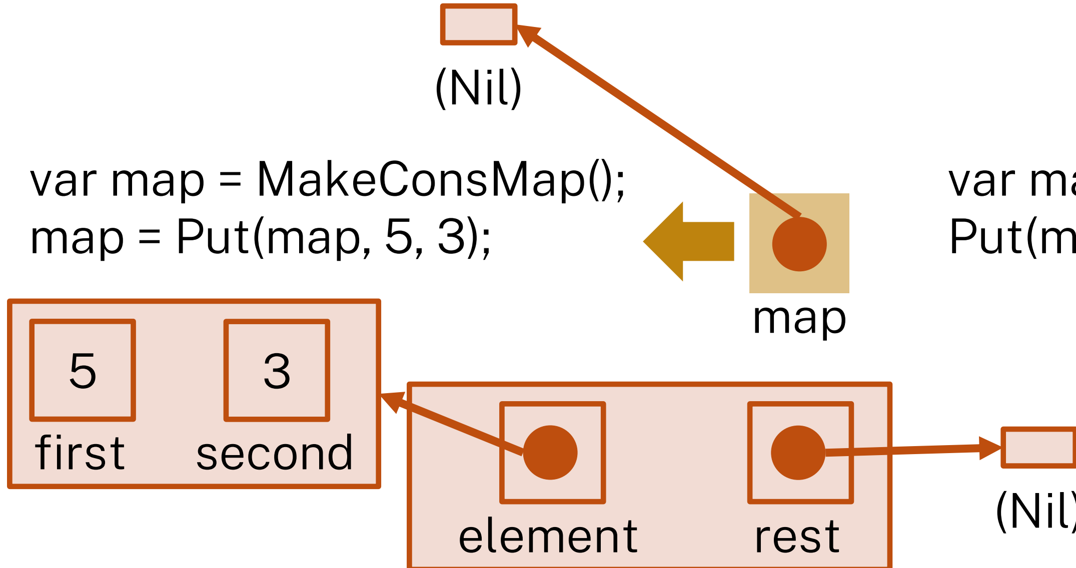


TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

Functional Programming

ConsList-based Map (CLM)

```
CLM Put(CLM, K, V)
CLM Remove(CLM, K)
```

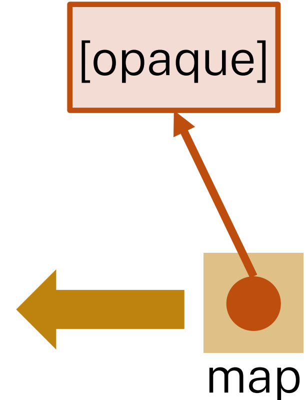


var map = MakeConsMap();
map = Put(map, 5, 3);

Imperative Programming

HashMap (HM)

```
void Put(HM, K, V)
void Remove(HM, K)
```



var map = MakeHashMap();
Put(map, 5, 3);

TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

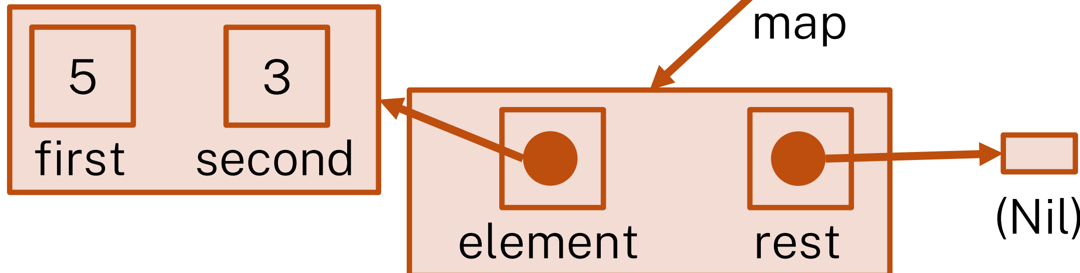
Functional Programming

ConsList-based Map (CLM)

```
CLM Put(CLM, K, V)
CLM Remove(CLM, K)
```

(Nil)

var map = MakeConsMap();
map = Put(map, 5, 3);

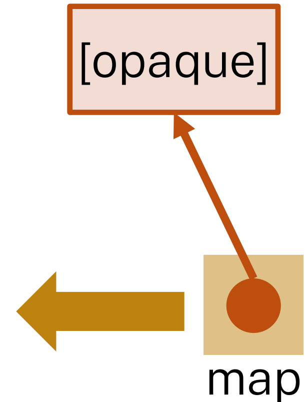


Imperative Programming

HashMap (HM)

```
void Put(HM, K, V)
void Remove(HM, K)
```

var map = MakeHashMap();
Put(map, 5, 3);



TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

The Design Recipe

With Mutable State

Australian National University

Problem Analysis and Data Design

1.) Figure out whether there should be mutable state in your data structure. Oftentimes, the answer is NO. Shared information or a need for memory are plausible reasons for YES.

2.) Be very explicit about which parts of your data might change, and how. Use invariants to express constraints on valid changes.

TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

Purpose Statement, Signature, *Effects*

Now that data can also flow via the heap, not just inputs and outputs, you need to describe what happens there. Add this between the purpose statement and the human signature (the “Design Strategy” in step 4 goes above this, between effects and examples).

```
int counter = 0;

/** …
 * Effect: increases the global counter by 1
 * … */
void incCounter() {
    counter = counter + 1;
}
```

Examples

Describe assumptions about relevant state in the example.

```
int counter = 0;
/** …
 * Examples:
 * Given: [nothing], and the global counter is 5
 * Expect: [nothing], and the global counter is 6
 * Effect: increases the global counter by 1
 * … */
void incCounter() {
    counter = counter + 1;
}
```

Design Strategy; Implementation

Largely the same, but implementation allows for new things (e.g. assignment).

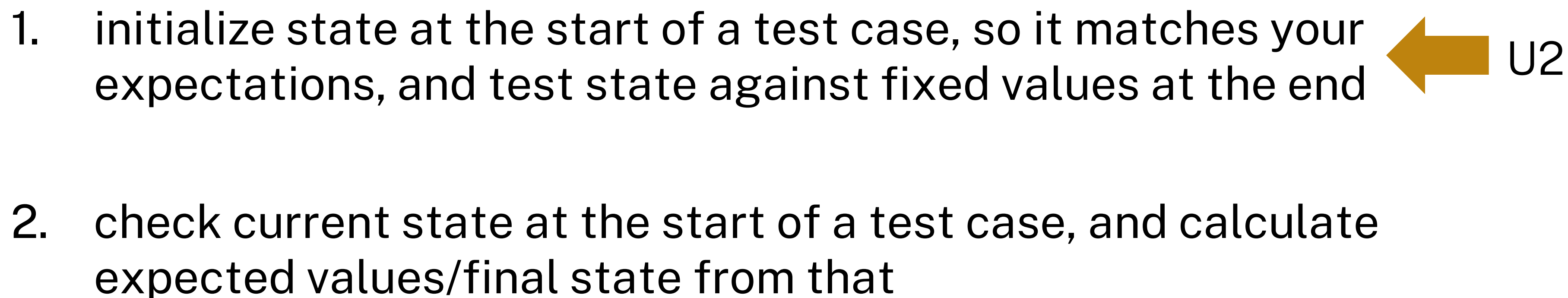
TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C

Tests

2 main options for dealing with state:

1. initialize state at the start of a test case, so it matches your expectations, and test state against fixed values at the end

2. check current state at the start of a test case, and calculate expected values/final state from that



TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C