THE UNIVERSITY OF MELBOURNE

COMP90041

Programming and Software Development

Tutorial 5 The Anatomy of Classes & Methods(CONT.)

Chuang(Frank) Wang

Slides were developed by Chuang Wang
Copyright @ The University of Melbourne

THE UNIVERSITY OF MELBOURNE

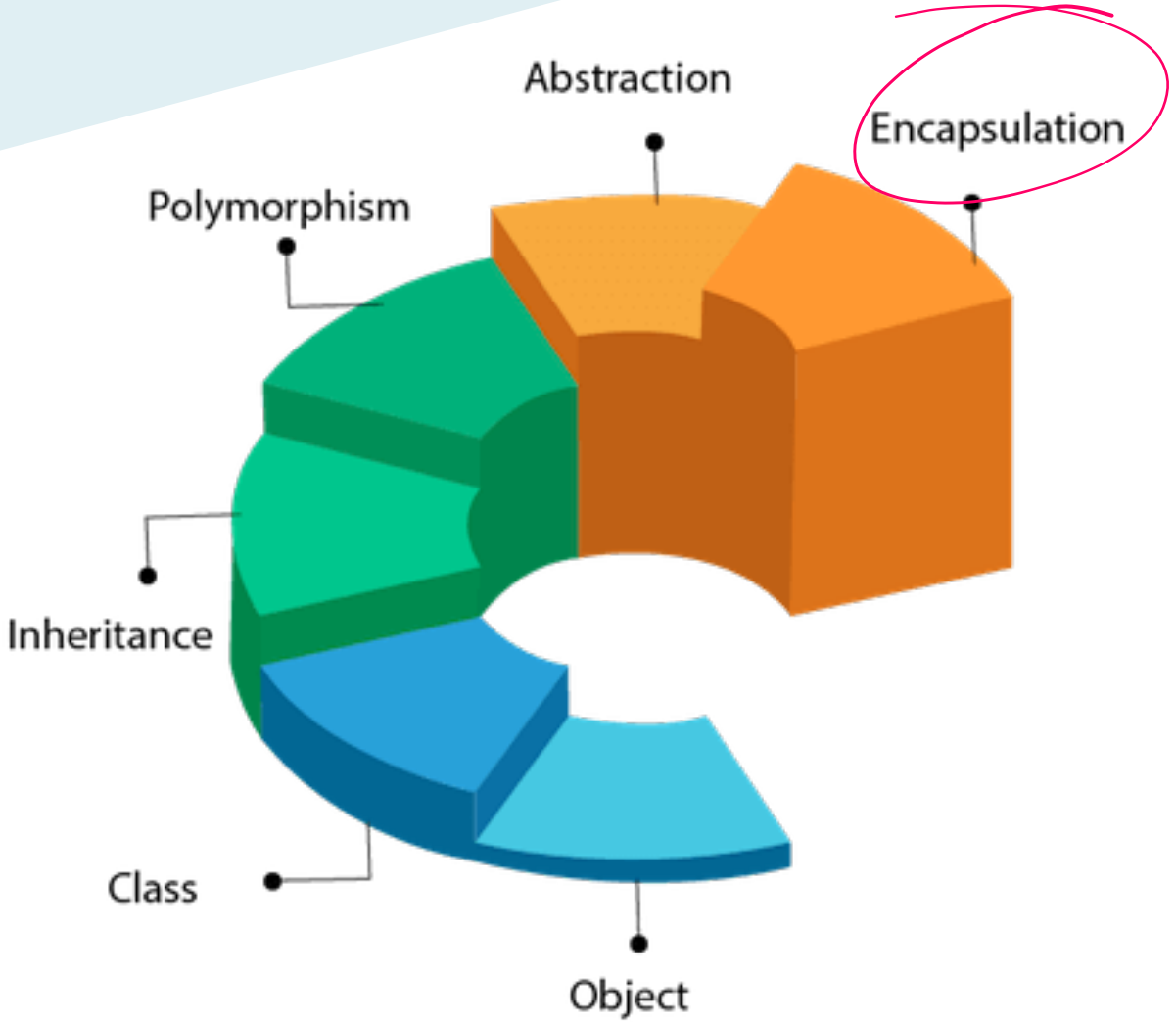
Overview

1. Classes & Objects
2. getter & setter & keywords
3. Immutable Objects
4. Exercise

THE UNIVERSITY OF
MELBOURNE

1. Classes & Objects

Encapsulation



- Short version:

  Data + Methods = Class(All details are hidden)

- Full version:

  Encapsulation means that the data and the actions are combined into a single item (in our case, a class object) and that the details of the implementation are hidden. Making all instance variables private is part of the encapsulation process.

Constructor

A special method that lives in the class instantiates an object(instance)

- ✓ Default constructor (automatically provided by Java)
  - Create an object only
- ✓ Your own constructor
  - Create an object
  - Initialize instance variables

Constructor(cont.)

Default

```
public Dog() {

}
```

```
Dog myDog = new Dog();
```

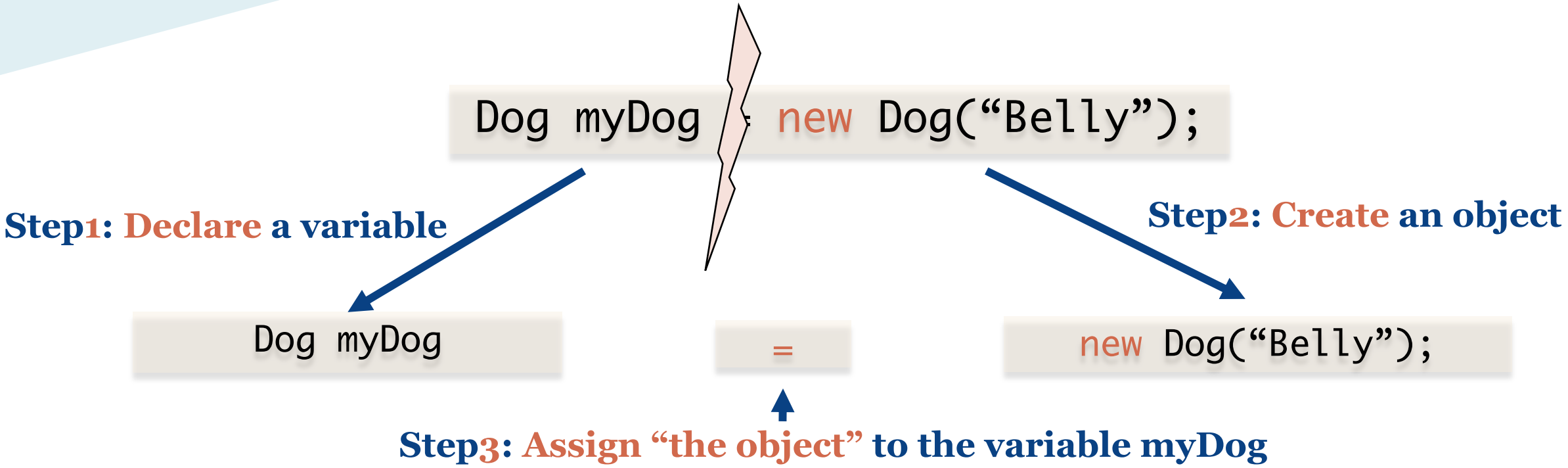
Your own

```
public Dog(String name, int age) {
      this.name = name;
      this.age = age;
}
```

```
Dog myDog = new Dog("Belly", 1);
```

Demo

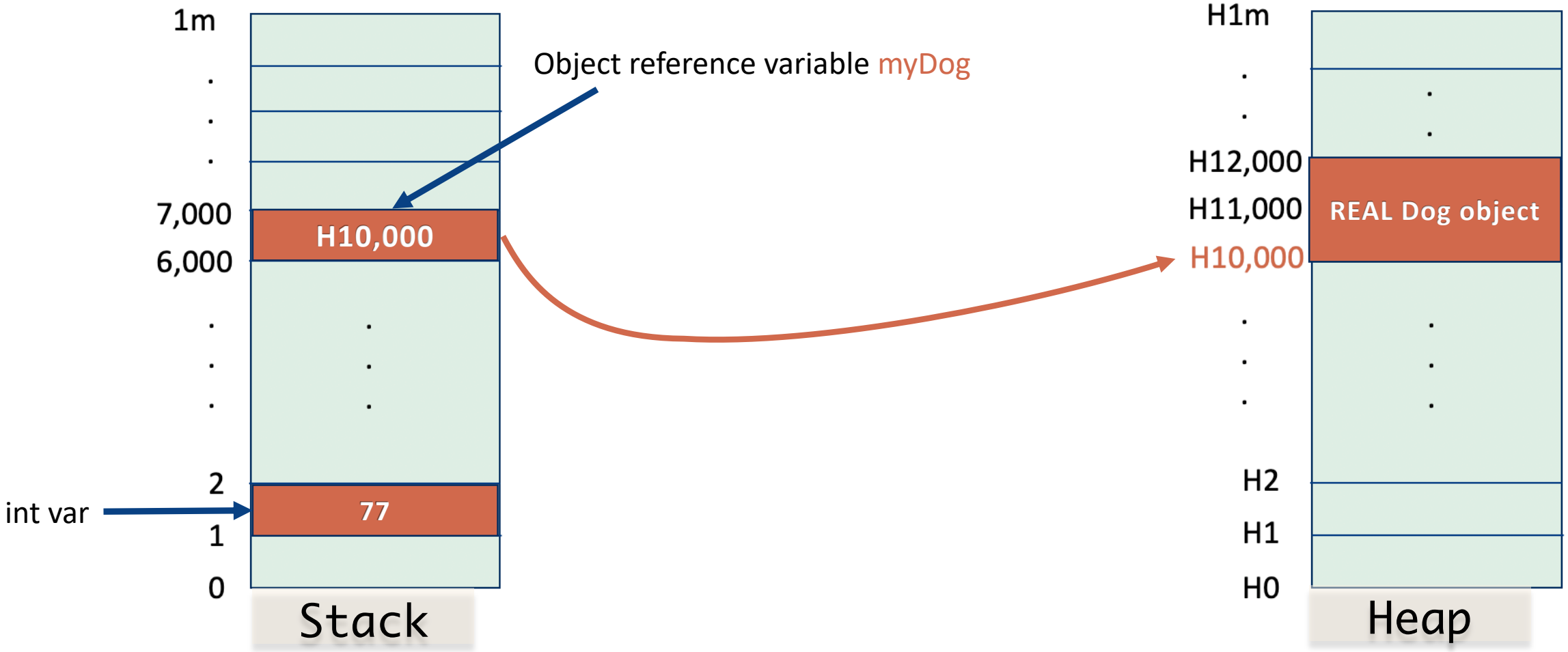
Create an object



Create an object(cont.)

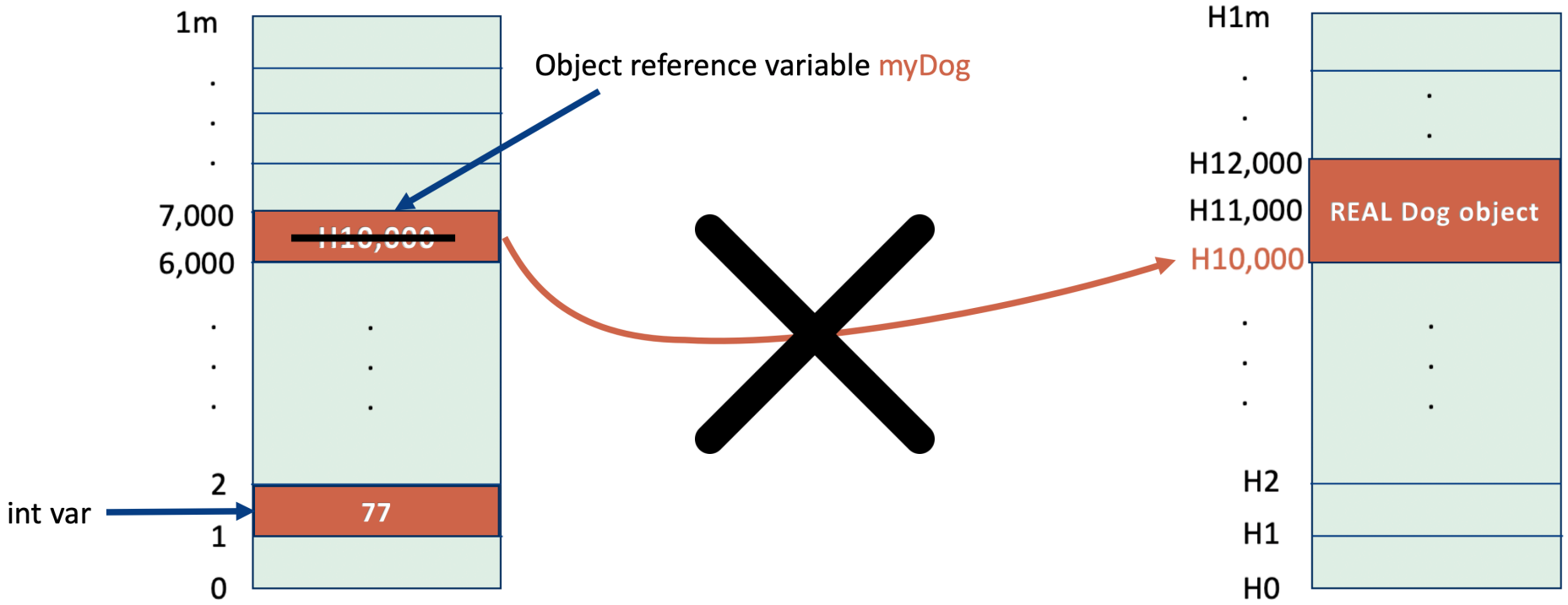
```
int var = 77;
```

```
Dog myDog = new Dog("Belly");
```



null

- ✓ A special constant
- ✓ A placeholder for a reference to an object variable.



THE UNIVERSITY OF
MELBOURNE

2. getter & setter

& keywords

Accessor VS Mutator

Why do we need them?

- `getter` – **Accessor**

  return the values of each instance variable

- `setter` – **Mutator**

  change the data in a class object

Accessor VS Mutator(cont.)

Why do we need them?

Why `private`?

force the users of those class to use methods to access them.

Why `getter`?

restrict access to read only(not write)

control the usage of private instance variables by adding checks(no plain access)

Why `setter`?

add some checks or conversions for the field access

Demo

this keyword

```
public Dog(String name, int age) {
        this.name = name;
        this.age = age;
}
```

- In a method

  Refer to the object that calls the method

- In a constructor

  Refer to the object being created by the constructor

final keyword

- prefix to an instance variable

  means the variable strictly cannot be changed

- In a method header(covered later)

  means the method cannot be redefined in a derived class

- In a class signature(covered later)

  means the class cannot be used as a base class to derive other classes.

`equals` and `toString`

- `equals`
  - compares the calling object to another object
  - should return true when the two objects are intuitively equal. When comparing objects of a class type, you normally use the method equals, not ==

- `toString`
  - return a string representation of the data in the calling object.
  - If a class has a `toString` method, you can use an object of the class as an argument to the methods `System.out.println`

Demo

THE UNIVERSITY OF
MELBOURNE

3. Immutable Objects

Immutable

- ✓ Immutable Object

  Cannot be changed once it is created

- ✓ Immutable Class

  is a class whose all instance(objects) are immutable

- ✓ String class & Primitive types are immutable

How to define immutable objects

- ✓ Make all fields final and private
- ✓ Don't provide "setter" methods
- ✓ Don't allow subclasses to override methods.

  (The simplest way to do this is to declare the class as final)
- ✓ If the instance fields include references to mutable objects, don't allow those objects to be changed

THE UNIVERSITY OF MELBOURNE

4. Exercise

Tutorial Exercise

1. Write an immutable `Movie` class to represent a theatrical movie. The class should have instance variables for the title, rank, and run time (in minutes). Give it the appropriate methods and constructors, but be sure to make the class immutable.

2. Add a `toString` method to the `Movie` class. The `toString` method should print the movie's rank and title in the same format as last lab.

3. Add an `equals` method to the `Movie` class. The `equals` method should return `true` if the both movie's title, rank and run time are equal, and `false` otherwise.

Homework

4. Write an immutable `Character` class to represent the characters in a movie. The class should have instance variables for the character's name, the name of the actor who played them, a rating, and the name of the movie the character appears in. Give it the appropriate methods and constructors, but be sure to make the class immutable.

   Revise your solution to Exercise 1 to include an instance variable of type `Character` for the main character of the movie.

THE UNIVERSITY OF MELBOURNE

Thank you

THE UNIVERSITY OF MELBOURNE

WARNING

This material has been reproduced and communicated to you by or on behalf of the University of Melbourne in accordance with section 113P of the *Copyright Act 1968* (**Act**).

The material in this communication may be subject to copyright under the Act.

Any further reproduction or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice