



COMP90041 Programming and Software Development

Tutorial 8 Inheritance

Chuang(Frank) Wang



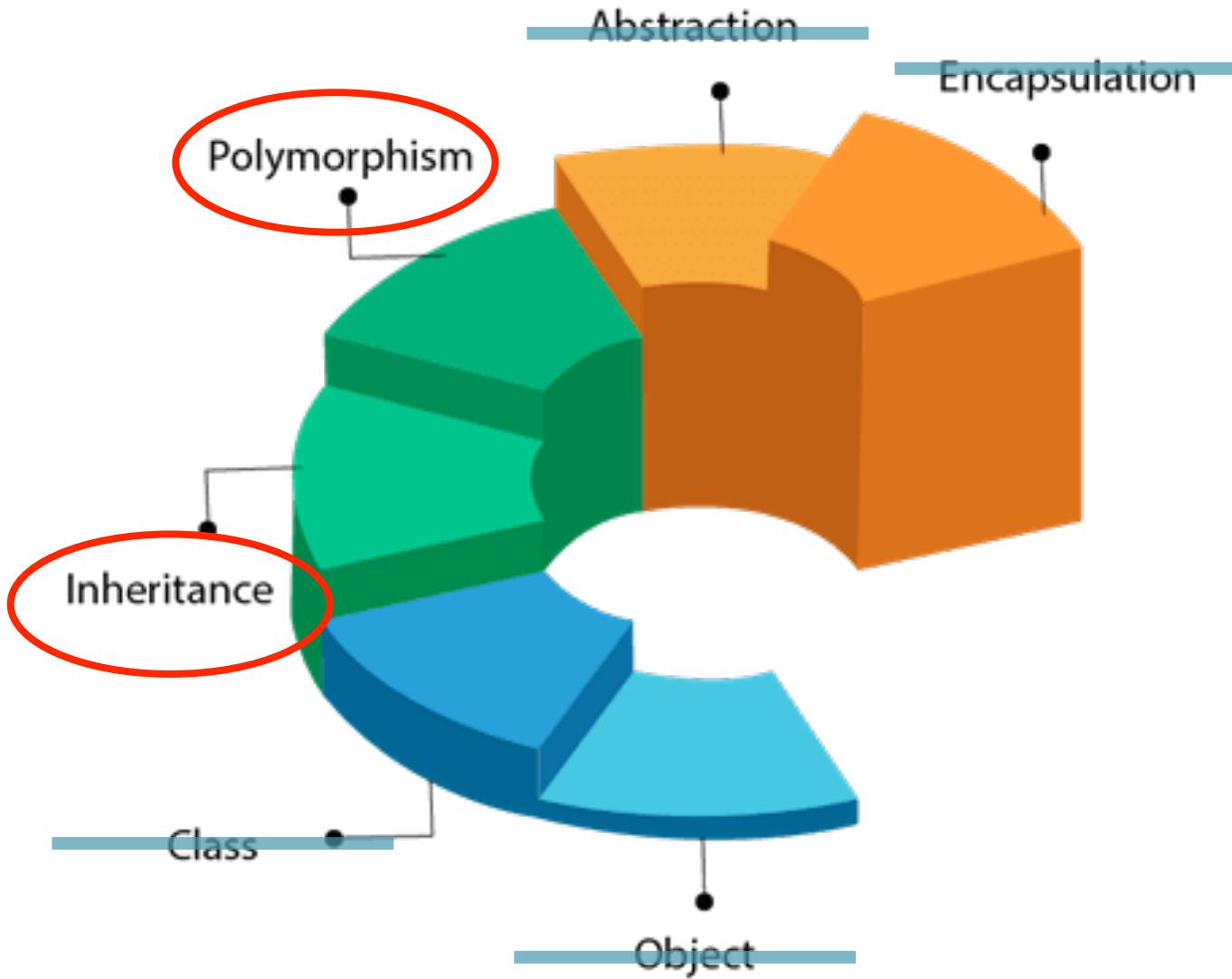


Overview

1. Type hierarchies(inheritance)
2. Method dispatch(late binding)
3. Exercise



OOP(object-oriented programming) Properties





THE UNIVERSITY OF
MELBOURNE

1. Type hierarchies (inheritance)



Types in Java

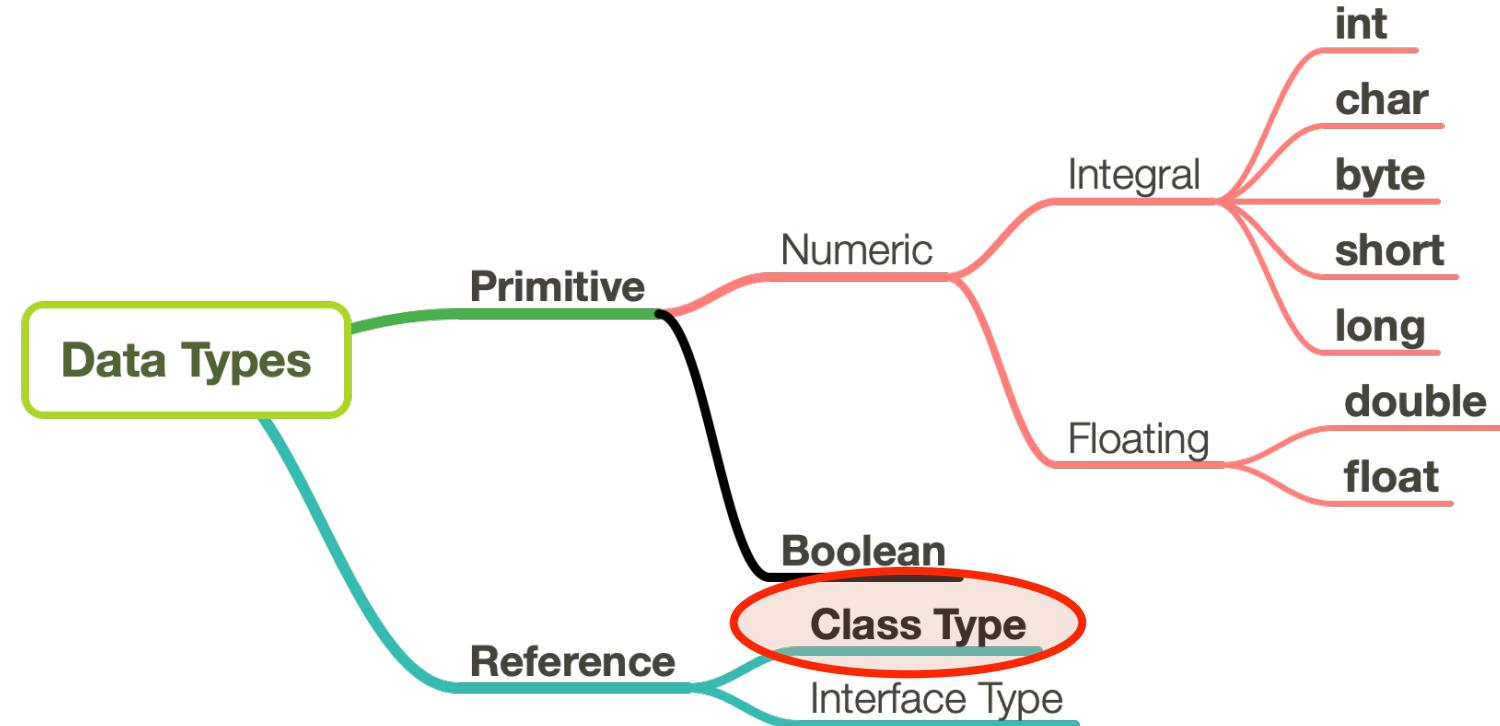
Java Class

1. Define a type

```
Employee e;
```

2. Create an object

```
e = new Employee();
```

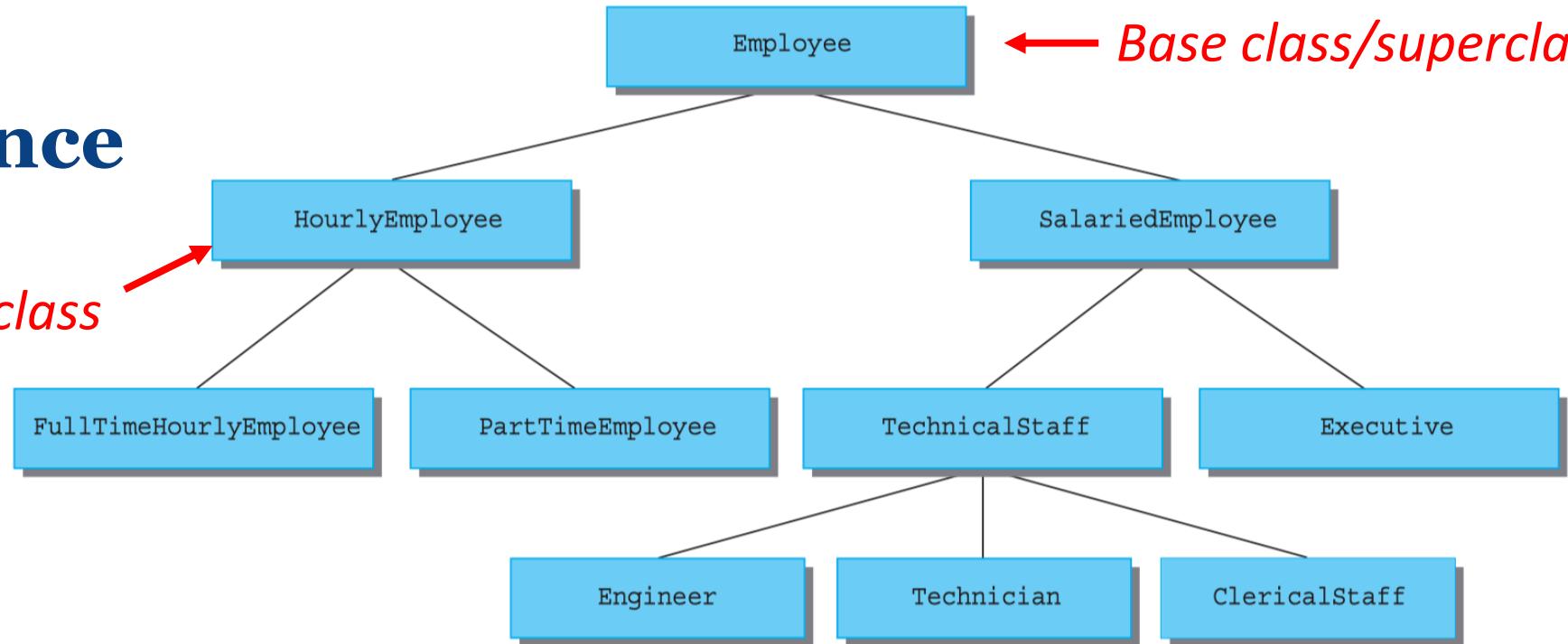




Inheritance

Derived class/subclass

← Base class/superclass



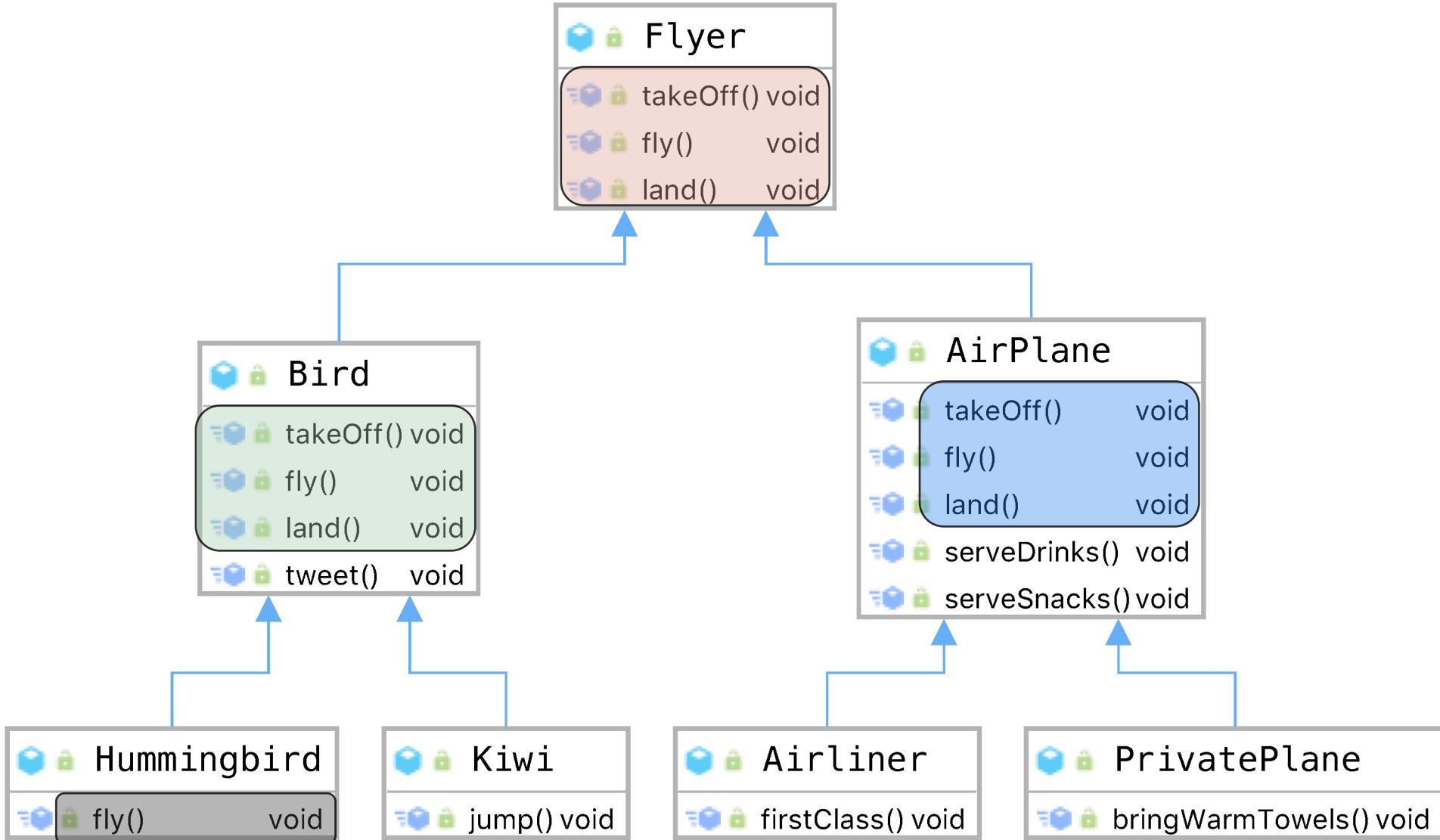
Shared attributes & methods!



- ✓ the ability of **building classes based on an existing class**
- ✓ so that they can have **inherited**(shared) data and methods, or further **adding/changing** methods



Another Example





Method Overriding VS Method Overloading

Method Overriding

a subclass can supply its **own implementation** for a method that also exists in the superclass.

```
public class Flyer {  
    ...  
    public void takeOff(){  
        System.out.println("takeoff from flyer class");  
    }  
    ...  
}
```

```
public class Bird extends Flyer{  
    ...  
    public void takeoff(){  
        System.out.println("takeoff from bird class");  
    }  
    ...  
}
```

Method Overloading

two methods have the **same name** but with different parameter lists
(only the types of the parameters matter)

```
public double computeGPA(double sub1, double sub2)  
public double computeGPA(int studentID)
```



THE UNIVERSITY OF
MELBOURNE

2. Method dispatch (late binding)



Method Dispatch

A subclass can

- ✓ Inherit all data & methods from its superclass
- ✓ Override methods that exist in superclass already
- ✓ Add extra methods in its own class

Method Dispatch

- ✓ Deciding which method to call (which implementation to use)



Apparent and actual types

```
Flyer f; //declare an object f
```

```
f = new Airplane(); // instantiate the object
```

```
Flyer f = new Airplane();
```



Declared/apparent type

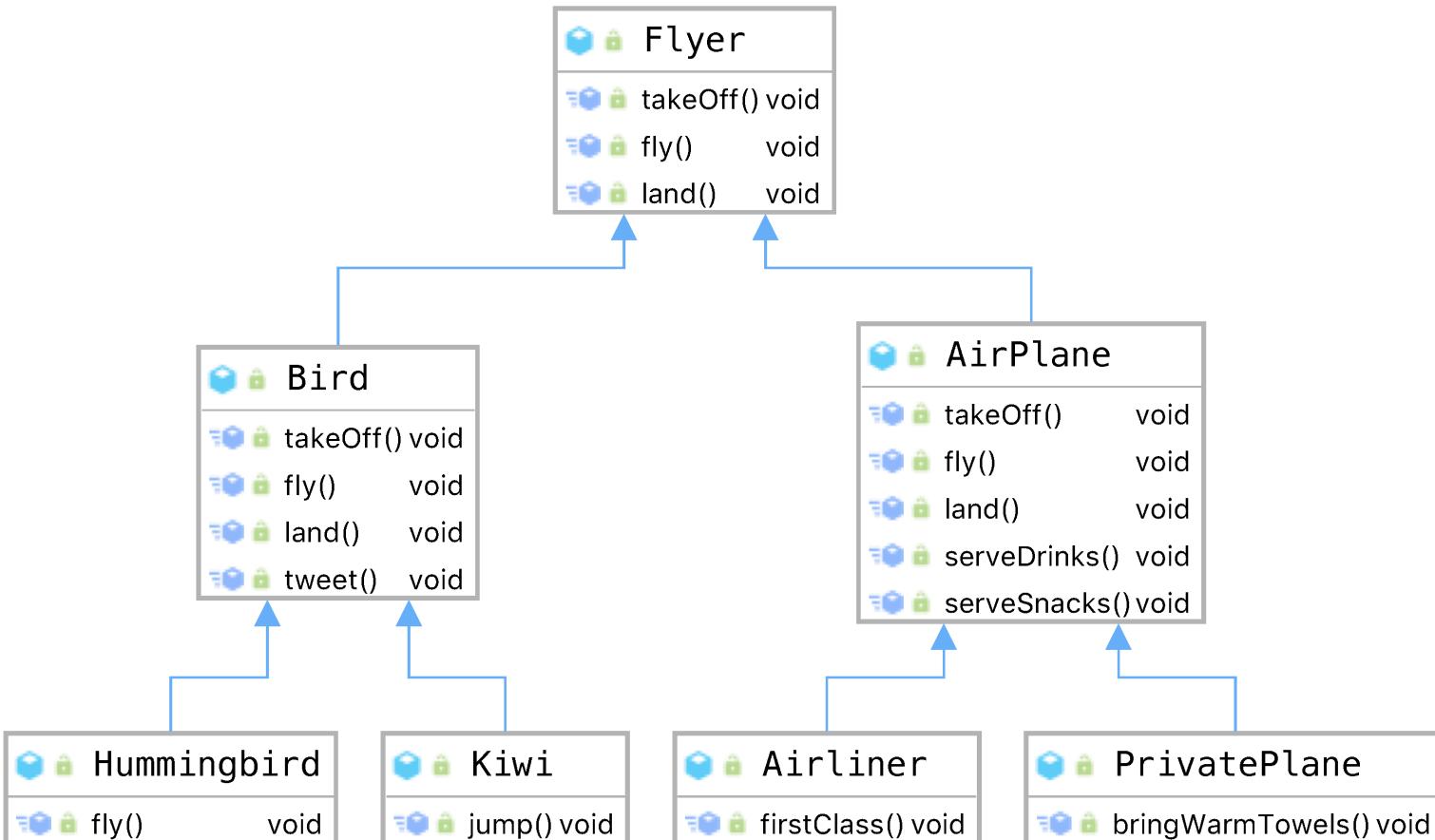
determines what the object can do
(what are methods available to the object)

actual/instantiated type

determines how the object will behave
(Which method implementation will be used)

*Java's ***method dispatch starts at the instantiated/actual type***, meaning that it first looks in the instantiated type for the implementation, and then looks up the hierarchy until it finds an implementation.

Questions



```

Flyer f = new Flyer();
Bird b = new Bird();
Kiwi k = new Kiwi();

PrivatePlane p = new PrivatePlane();

// which of the below statements are valid?

1. f.fly(); ✓
2. f.bringWarmTowels(); ✗
3. f(tweet()); ✗
4. b.takeOff(); ✓
5. b.jump(); ✗
6. k.jump(); ✓
7. k(tweet()); ✓
8. p.serveDrinks(); ✓
9. p.bringWarmTowels(); ✓
  
```

The code snippet at the top defines objects of the **Flyer**, **Bird**, **Kiwi**, and **PrivatePlane** classes. Below this, a series of numbered statements are presented, each followed by a green checkmark or a red X. The statements are:

- `f.fly();` ✓
- `f.bringWarmTowels();` ✗
- `f(tweet());` ✗
- `b.takeOff();` ✓
- `b.jump();` ✗
- `k.jump();` ✓
- `k(tweet());` ✓
- `p.serveDrinks();` ✓
- `p.bringWarmTowels();` ✓



Questions

```
public class Flyer {  
    ...  
    public void takeOff(){  
        System.out.println("takeoff from flyer class");  
    }  
    ...  
}
```

```
public class Bird extends Flyer{  
    ...  
    public void takeoff(){  
        System.out.println("takeoff from bird class");  
    }  
    ...  
}
```

What is the output?

```
Flyer f = new Bird();  
f.takeoff();
```



THE UNIVERSITY OF
MELBOURNE

3. Exercise



Tutorial Exercise

1. Write a java **Shoe** class. Every shoe should have a color (String), designer (String) and size (float). Your class should provide all the usual methods (i.e. constructors, accessors and mutators).
2. Write a java **DressShoe** class. In addition to the attributes listed above, a dress shoe should have a heel type that is one of: pump, heel, or flat.



Homework

3. Write a java **TennisShoe** class. In addition to the attributes listed for **Shoe**, tennis shoes should have a sole type and canvas type, both Strings.
4. Write a java **Boot** class. In addition to the attributes listed for **Shoe**, boots should have a heel type that is one of: pump, heel, or flat.
5. Add a **toString** and **equals** methods to each of these classes.
6. Define an enum to represent heel type, ensuring that only valid heel types are used in both the **DressShoe** and **Boot** classes. Where no heel type is known, assume it is a heel.

Hint: you can define an enum type in a separate file much as you would define a class. Just use the word **enum** in place of **class**, and list the enum constants between the braces instead of instance variables and methods.



Thank you





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