Victoria University of Wellington, New Zealand
Te Whare Wananga o te Upoko o te Ika a Maui Aotearoa

SWEN221: Software Development

7: Inheritance I

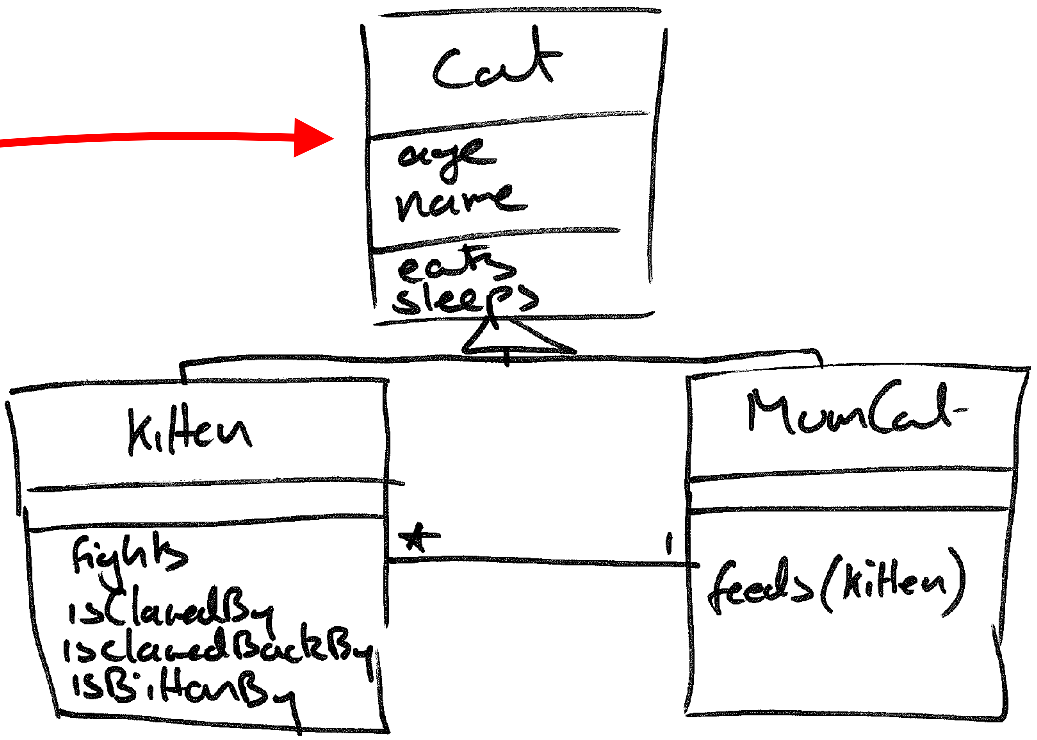
David J. Pearce & Marco Servetto
Computer Science, Victoria University

Inheritance basics

superclass

& baseclass

subclass



- Kitten & MumCat
  - Inherit attributes “age” and “name”
  - Inherit operations “eats” and “sleeps”

Inheritance basics

```
class Cat {
    int age;
    …
}
class Kitten extends Cat {
    void fights() {…}
    …
}
class MumCat extends Cat {…}
```

Inheritance

- What does it give us?

Subtyping

&

Code Reuse

What is Subtyping?

- In Java, can write the following:

```
void g(Object y) { … }
void f(String x) { g(x); }
```

  - This is OK because a String is always a valid Object
  - But, this **does not** compile:

```
void g(String y) { … }
void f(Object x) { g(x); }
```

  - Because an Object is not always a valid String.
    - E.g. an ArrayList is not a String

- We say String **is a subtype of** Object
  - denoted by **`String <: Object`**
  - A subtype can be used whenever its supertype(s) are expected

Inheritance and Subtyping

- For two classes/interfaces A and B:
  - if A **extends** B, or A **implements** B, then A `<:` B

```
class Point { int xpos; int ypos; … }
class ColouredPoint extends Point { int colour; }

void move(Point p, int dx, int dy) {
 p.xpos += dx;
 p.ypos += dy;
}
ColouredPoint cp = new ColouredPoint(…);
move(cp,1,1);
System.out.println("cp.xpos = " + cp.xpos);
```

Through p we cannot see "colour" but it is there!

  - Therefore, in this code, ColouredPoint `<:` Point
  - Meaning we can use a ColouredPoint instead of a Point!

Static vs Dynamic Typing

- **What is it?**
  - **Static Type** – the declared type of a variable in the code
  - **Dynamic Type** – the actual type of an object in a moment of the execution

```
class Point { int xpos; int ypos; … }
class ColouredPoint extends Point { int colour; }

void move(Point p, int dx, int dy) {
 p.xpos += dx;
 p.ypos += dy;
}
move(new ColouredPoint(…),1,1);
System.out.println(''cp.xpos = '' + cp.xpos);
```

  - Here, parameter p has **static type** Point
  - But, in the shown call, p refers to object with **dynamic type** ColouredPoint
  - Can only access fields/methods through static type of p

Properties of Subtyping

- Subtyping properties:
  - Transitive
    - If X <: Y and Y <: Z then X <: Z
  - Reflexive
    - X <: X always holds!

```
class Point { int xpos; int ypos; … }
class ColouredPoint extends Point { int colour; }
class Coloured3DPoint extends ColouredPoint { int z; }
```

- So, does Coloured3DPoint <: Point hold?

Exercise – which ones work?

```
class Point { int xpos; int ypos; … }
class Point3D extends Point { int z; }
class ColouredPoint extends Point { int colour; }
class ColouredPoint3D extends ColouredPoint { int z; }

void move(Point p) { … }
void paint(ColouredPoint cp) { … }

ColouredPoint3D cp3 = new ColouredPoint3D(…);
Point3D p3 = new Point3D(…);
```

A) `move(cp3);` B) `move(p3);` C) `paint(p3);`

Exercise – which ones work?

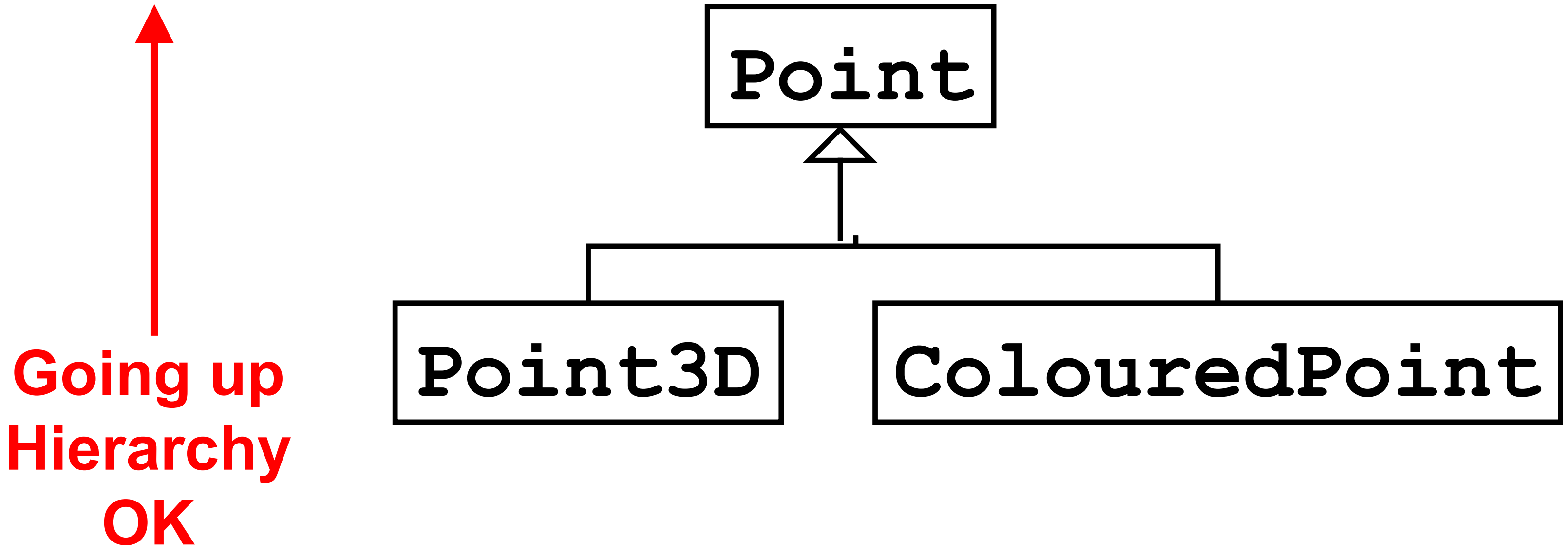
```
class Point { int xpos; int ypos; … }
class Point3D extends Point { int z; }
class ColouredPoint extends Point { int colour; }
class ColouredPoint3D extends ColouredPoint { int z; }

void move(Point p) { … }
void paint(ColouredPoint cp) { … }

ColouredPoint3D cp3 = new ColouredPoint3D(…);
Point3D p3 = new Point3D(…);
```

A) [✓ move(cp3);] B) [✓ move(p3);] C) paint(p3); ✗

Up Casting

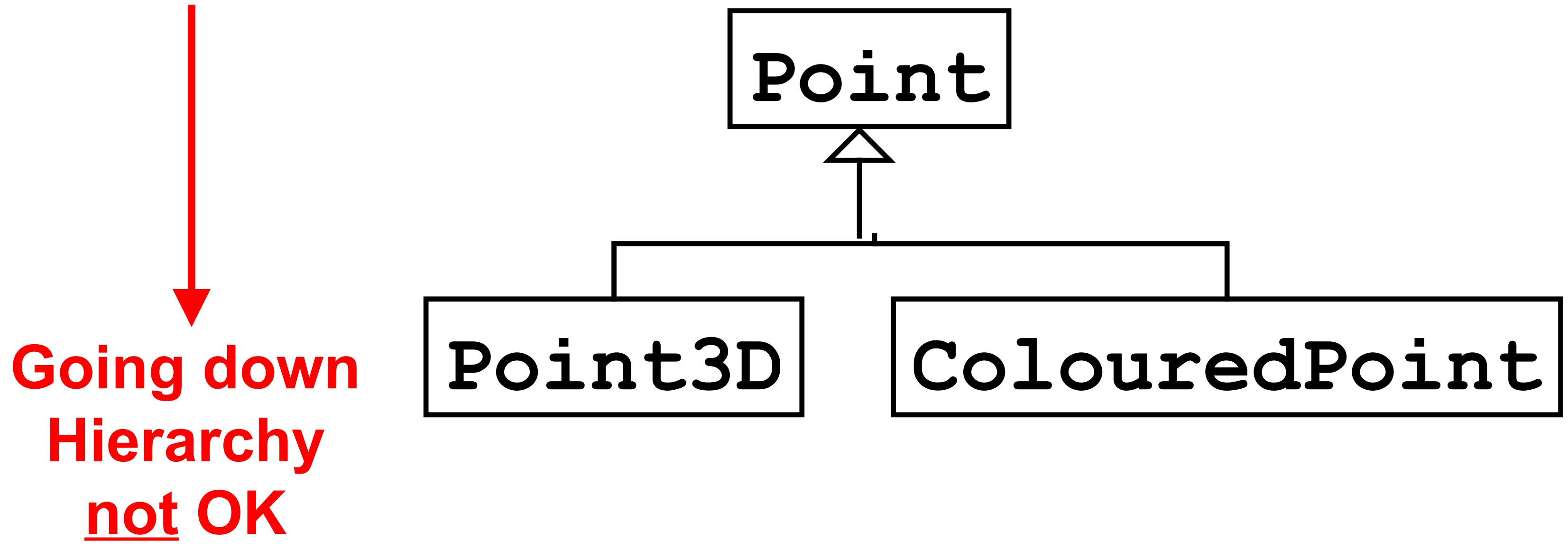


```
Point3D doSomething() { … }

Point p = doSomething();
```

✓

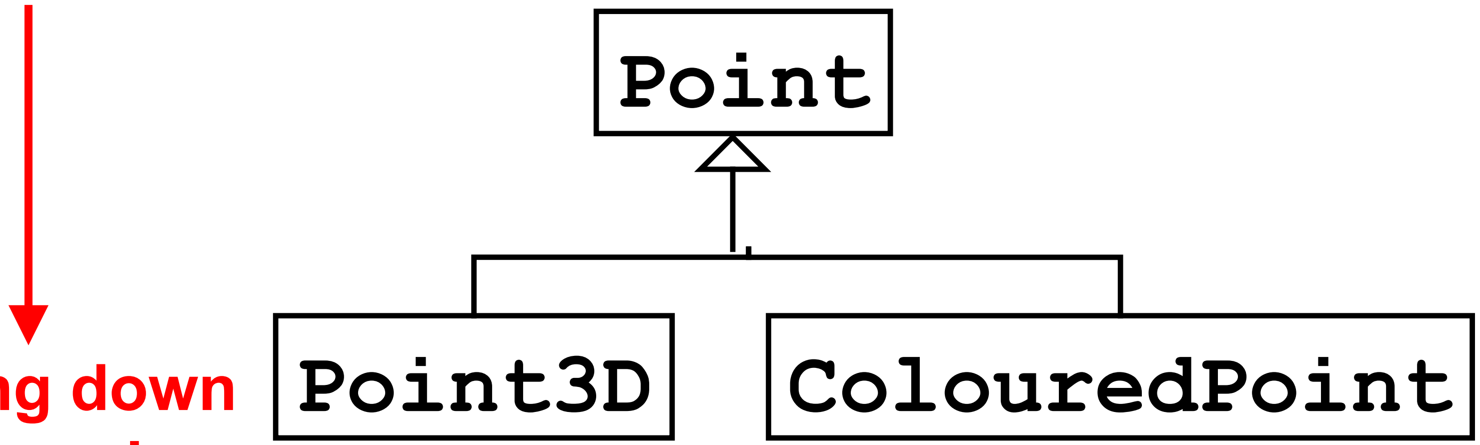
Down Casting



```
Point doSomething() { … }


Point3D p = doSomething();
```

Down Casting



```
Point doSomething() { … }


Point3D p = (Point3D) doSomething();
```

✓

- Will throw exception if not Point3D!

Inheritance + Method overriding

- Can **override** methods of superclass:

```
class A {
 void aMethod() {
  System.out.println("A called");
}}
class B extends A {
 void aMethod() {
  System.out.println("B called");
}}
A x = new A();
A y = new B();
x.aMethod();
y.aMethod();
```

B.aMethod()
overrides
A.aMethod()

Static vs Dynamic Typing (again)

```
...

A x = new A();  // static type of x is A, dynamic type of x is A
A y = new B();  // static type of y is A, dynamic type of y is B
x.aMethod();
y.aMethod();
```

- Static Type
  - Types written in the **program source**
  - Every variable or field has a **static type**
- Dynamic Type
  - **Actual type** of an object
  - May be **different** from static type, may change during execution
  - Determined when object **created** using new
  - Dynamic type of variable always **subtype** of static type

Quiz – what gets printed?

```
class Person { … }

class Car {
 void shutDoor(Person p) {
  System.out.println("Door shuts");
}}
class BigCar extends Car {
 void shutDoor(Person p) {
  System.out.println("Door SLAMS!");
}}

Car c = new Car();
BigCar b = new BigCar();
Person jim = new Person();
c.shutDoor(jim);
b.shutDoor(jim);
```

A)

"Door shuts"

"Door shuts"

B)

"Door SLAMS!"

"Door SLAMS!"

C)

"Door shuts"

"Door SLAMS!"

Quiz – what gets printed?

```
class Person { … }

class Car {
 void shutDoor(Person p) {
  System.out.println(“Door shuts”);
}}
class BigCar extends Car {
 void shutDoor(Person p) {
  System.out.println(“Door SLAMS!”);
}}

Car c = new Car();
BigCar b = new BigCar();
Person jim = new Person();
c.shutDoor(jim);
b.shutDoor(jim);
```

A)
“Door shuts”
“Door shuts”

B)
“Door SLAMS!”
“Door SLAMS!”

[✓ C)
“Door shuts”
“Door SLAMS!”]

Quiz – what gets printed?

```
class Person { … }

class Car {
 void shutDoor(Person p) {
  System.out.println("Door shuts");
}}
class BigCar extends Car {
 void shutDoor(Person p) {
  System.out.println("Door SLAMS!");
}}

Car c1 = new Car();
Car c2 = new BigCar();
Person jim = new Person();
c1.shutDoor(jim);
c2.shutDoor(jim);
```

A)
"Door shuts"
"Door shuts"

B)
"Door SLAMS!"
"Door SLAMS!"

C)
"Door shuts"
"Door SLAMS!"

Quiz – what gets printed?

```
class Person { … }

class Car {
 void shutDoor(Person p) {
  System.out.println("Door shuts");
}}
class BigCar extends Car {
 void shutDoor(Person p) {
  System.out.println("Door SLAMS!");
}}

Car c1 = new Car();
Car c2 = new BigCar();
Person jim = new Person();
c1.shutDoor(jim);
c2.shutDoor(jim);
```

A)
"Door shuts"
"Door shuts"

B)
"Door SLAMS!"
"Door SLAMS!"

[✓ C)
"Door shuts"
"Door SLAMS!"]