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

SWEN221 Software Development

Inheritance I

Thomas Kuehne

Victoria University

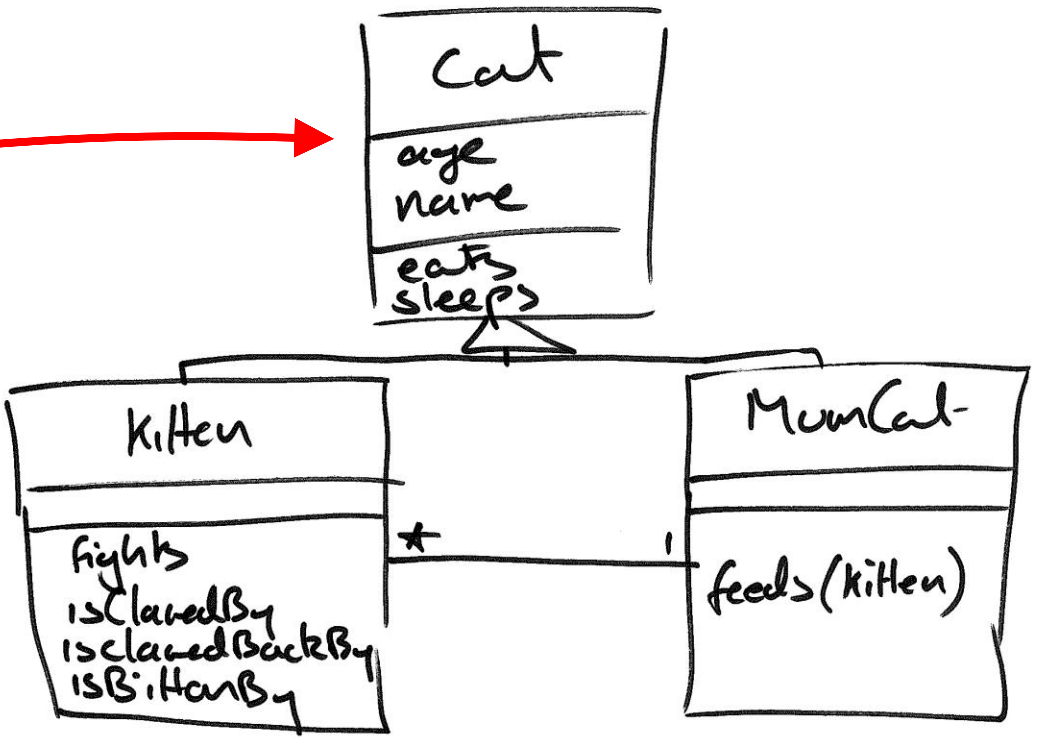
(slides modified from slides by David J. Pearce & Nicholas Cameron & James Noble & Petra Malik)

Inheritance basics

superclass

(aka 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:

```
Cat c;
c = new Kitten();
```

  - This is OK because a kitten can be used in place of a cat
  - But, the following **does not** compile:

```
Kitten k;
k = new Cat();
```

  - because a cat cannot be used in place of a kitten.
    - a cat does not exhibit the behaviour expected of a kitten

- We say **`Kitten` is a subtype of `Cat`**
  - denoted by **`Kitten <: Cat`**
  - Instances of a subtype can be used whenever instances of 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 moveBy(Point p, int dx, int dy) {
 p.xpos += dx;
 p.ypos += dy;
}
ColouredPoint cp = new ColouredPoint(…);
moveBy(cp, 1, 1);
System.out.println("cp.xpos = " + cp.xpos);
```

Through **p** we cannot see "colour" but the object still has the field!

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

Static vs Dynamic Types

- **Static Type** – the (never changing) declared type of a variable
- **Dynamic Type** – the (potentially changing) actual type of an object referenced by the variable

```
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);
```

  – Here, parameter **p** has **static type** Point
  – But, **p** refers to object with **dynamic type** ColouredPoint
  – Can only access fields/methods exposed by static type of **p**
  – Behaviour is determined by the dynamic type of **p**, though

Properties of Subtyping

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

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

- Hence, Coloured3DPoint <: Point

Exercise – which ones work?

```
class Point { int xpos; int ypos; … }
class 3DPoint extends Point { int z; }
class ColouredPoint extends Point { int colour; }
class Coloured3DPoint extends ColouredPoint { int z; }

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

Coloured3DPoint c3p = new Coloured3DPoint(…);
3DPoint 3p = new 3DPoint(…);
```

A) `move(c3p);` B) `move(3p);` C) `paint(3p);`

Inheritance + Method overriding

- superclass methods can be **overridden**

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

- Static Type
  - Types written in the **program source**
  - Every variable or field has a **static type**

Static vs Dynamic Typing (cont'd)

```
…

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

- Dynamic Type
  - **Actual type** of an object referenced by a variable
  - May be **different** from static type
  - Determined when object is assigned to variable
  - Dynamic type of variable is always a **subtype** of its static type

Quiz – what gets printed?

```
class Car {
 void shutDoor() {
  System.out.println("Door shuts.");
}}

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


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

A)

"Door shuts."

"Door shuts."

B)

"Door SLAMS!"

"Door SLAMS!"

C)

"Door shuts."

"Door SLAMS!"

Inheritance + Method overriding

- Can access overridden methods via "`super`"

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


   System.out.println("B called");
}}

B b = new B();
b.aMethod(); // prints: " A called
                         B called"
```

Inheritance + Constructors

- "Super" can also be used in a constructor to access the superclass constructor:

```
class A {
 A(Object aParam) {…}
}}

class B extends A {
  B(Object aParam, Object anotherParam) {
  super(aParam);
  …
}}
```