Australian National University

J2

Introductory Java 2

Types

Objects

Classes

Inheritance

Interfaces

Types

`00110001 ?`

$2^5+2^4+2^0 = 49$?

`R of RGB value ?`

`ASCII Char '1' ?`

`X86 Opcode XOR ?`

Type-Soundness: at most one of those interpretations is true for any concrete sequence of bits.

5 + 4 ✓

"Hello" % 3 ✗

Static Type-Checking: ensure type-safety before running the program (e.g. Java)

Dynamic Type-Checking: crash instead of doing non-sensical operations (e.g. Python)

String x; ✗

int x; ✓

x % 3 ?

Objects

State ✚ Behavior

Fields (with Types)

Methods (imperative code)

Good OO code guards state with behavior

Example: Bicycle

(Current)

- Speed
- Direction
- Cadence
- Gear

- Change Gear
- Change Cadence
- Brake

Java Interfaces

Methods define behavior

An **`interface`** is a group of methods *without* implementations

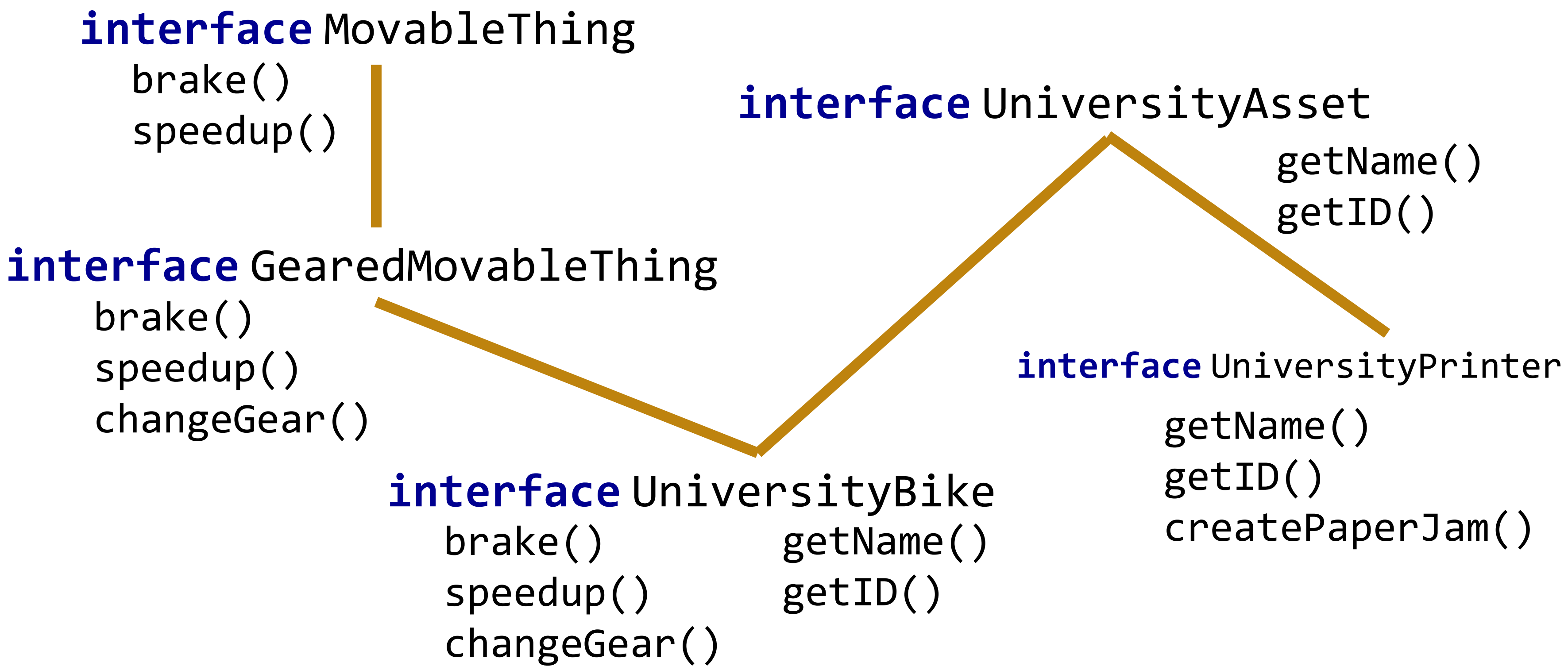
Example: an **`interface`** `MovableThing` might include:

```
 brake()
 speedup()
```

Interfaces can **extend** other interfaces, for example, an **`interface`** `GearedMovableThing` is a `MovableThing` that also includes:

```
 changeGear()
```

Interface Hierarchy



Classes

Describe the state and behavior of similar objects – a “blueprint” from which to create objects.

Are the **most precise type** of objects created from them.

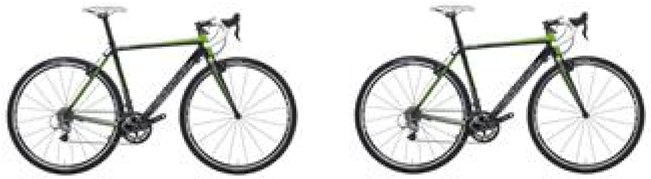
Vice versa: **Objects are instances of Classes.**

Example: bicycle

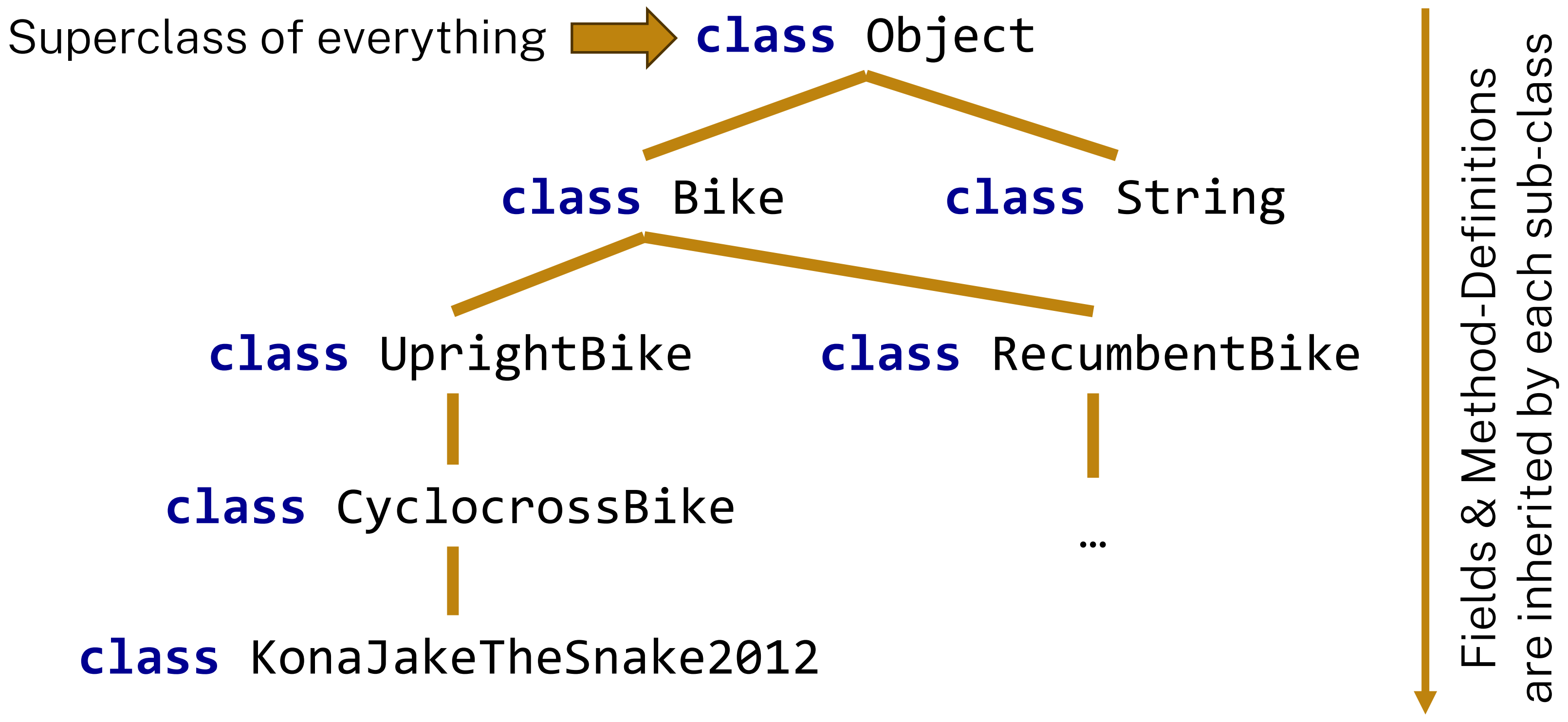
Class: Kona Jake The Snake 2012

Instance: your bike

Another Instance: my bike



Class Hierarchy: Inheritance



Key Java OO Concepts Overview

Types

Interface	**Class**	**Object**
Describes behavior, but not state	Blueprint, describes both state and behavior, including implementation	Concrete instance of a class, specific state
No implementation		
Useful to guard your objects' internals - **"Encapsulation"**	May implement interfaces	A **"value"** in your program, like 5 or the String "Hello"