Lab6（week8）

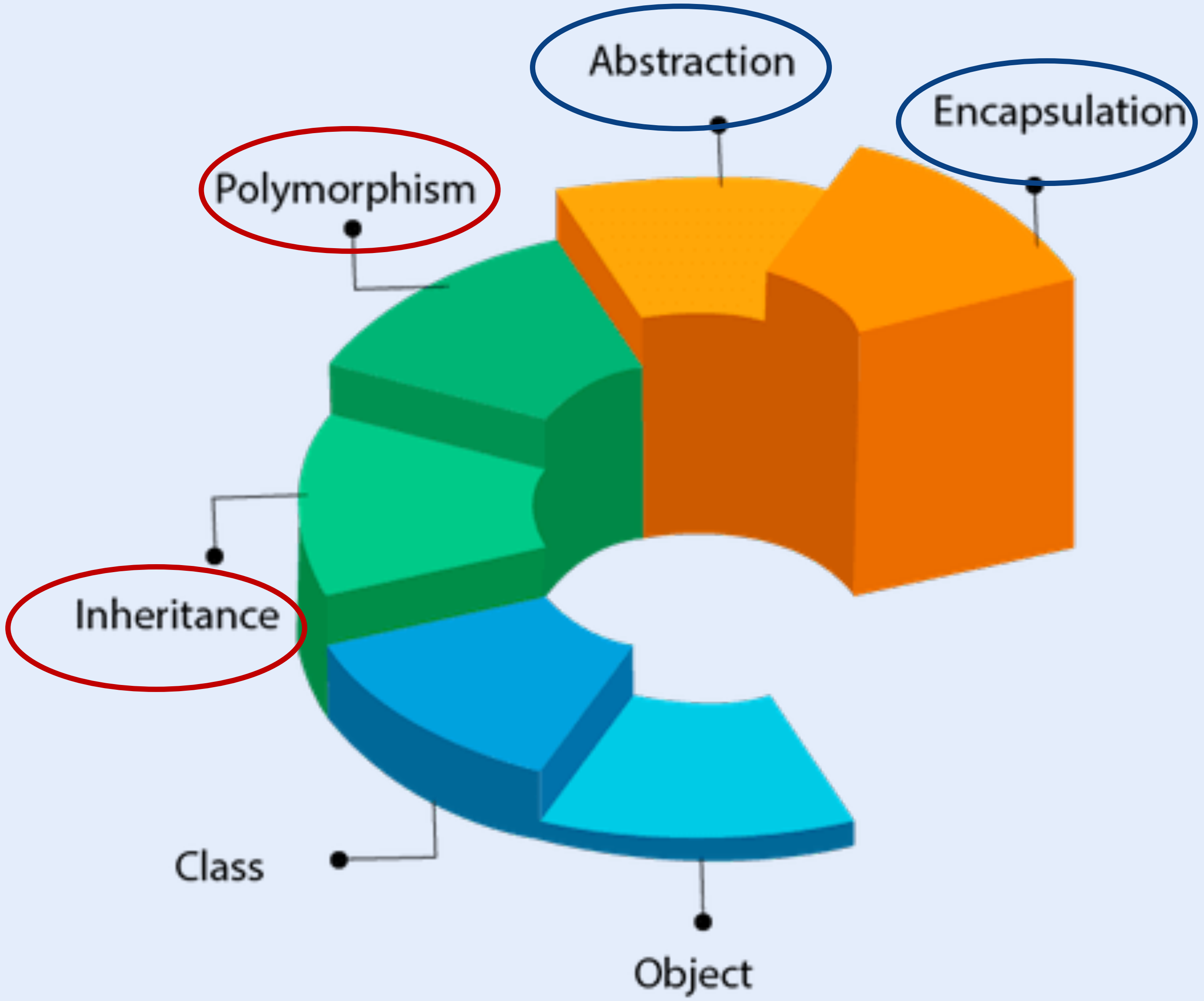
COMP90041 Programming

and software development

Zhe(Zoe) Wang

github: https://github.com/Zoeewang/COMP90041-2020-sem1-tutorial

OOPs (Object-Oriented Programming System)

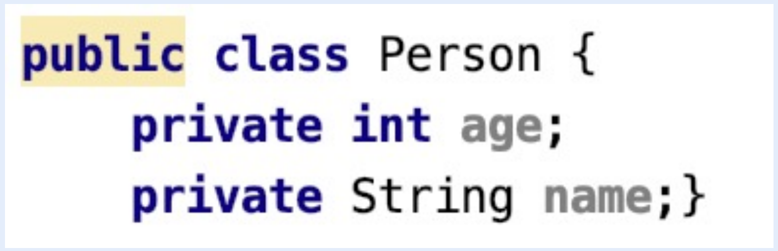


inheritance

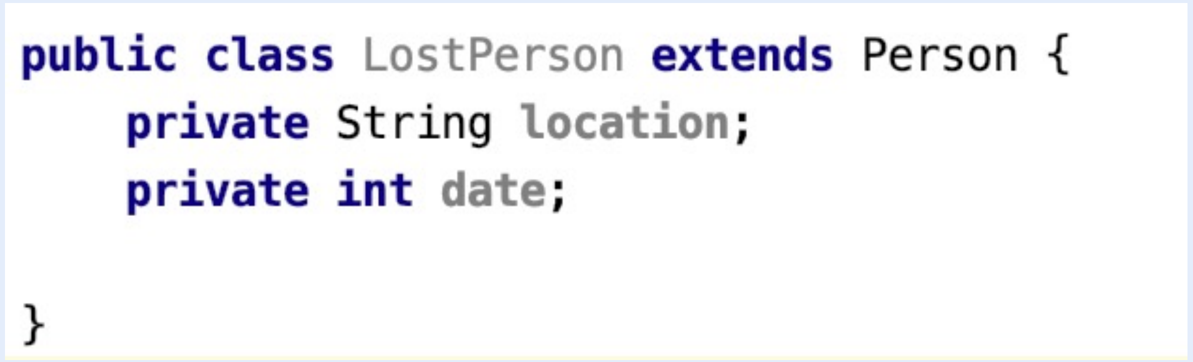
subclass / child class

inheritance allows a derived class to be defined by specifying only how it differs from base class superclass / parent class

form: extends BaseClass{ …. }



```
public class Person {
    private int age;
    private String name;}
```



```
public class LostPerson extends Person {
    private String location;
    private int date;


}
```

LostPerson class inherits all the instance variables and methods of the Person class….and adds its own!

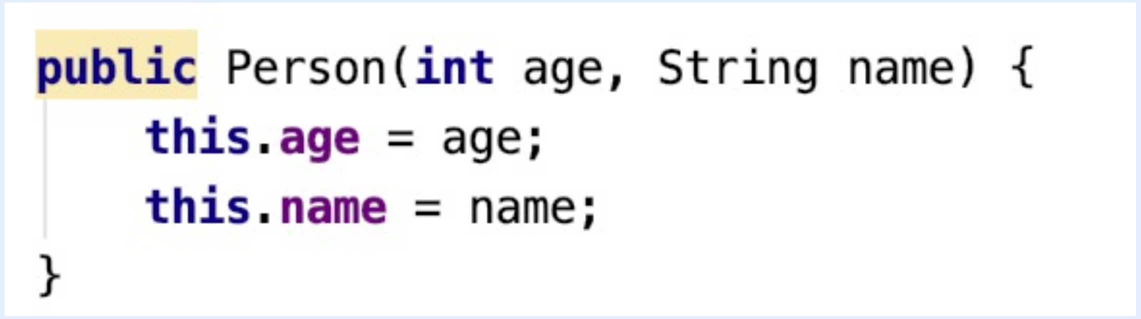
No need to mention inherited instance variables and methods

super Constructor

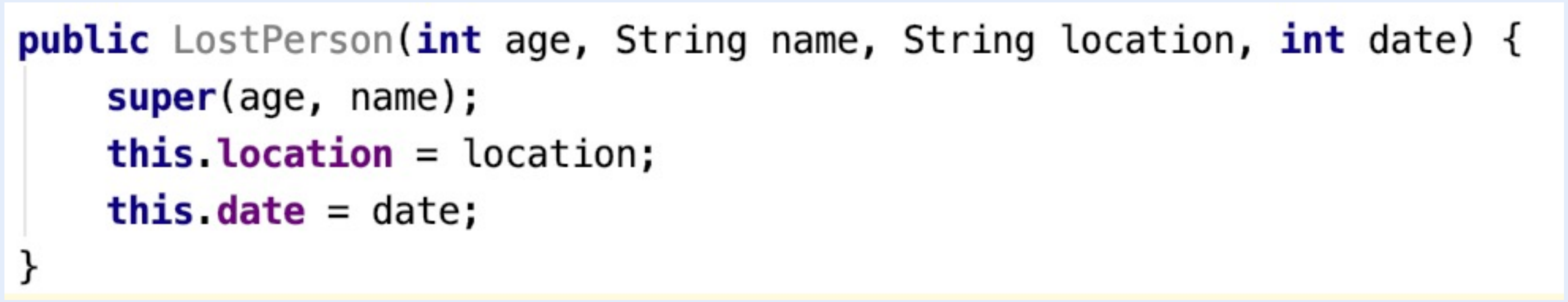
Constructors are not inherited, cannot be overridden !

Constructor chaining: derived class constructor must invoke base class constructor first.

form: super(arguments…)



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

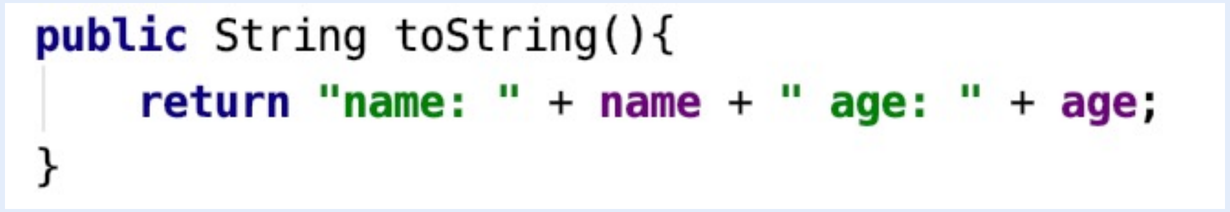


```
public LostPerson(int age, String name, String location, int date) {
    super(age, name);
    this.location = location;
    this.date = date;
}
```

Overriding

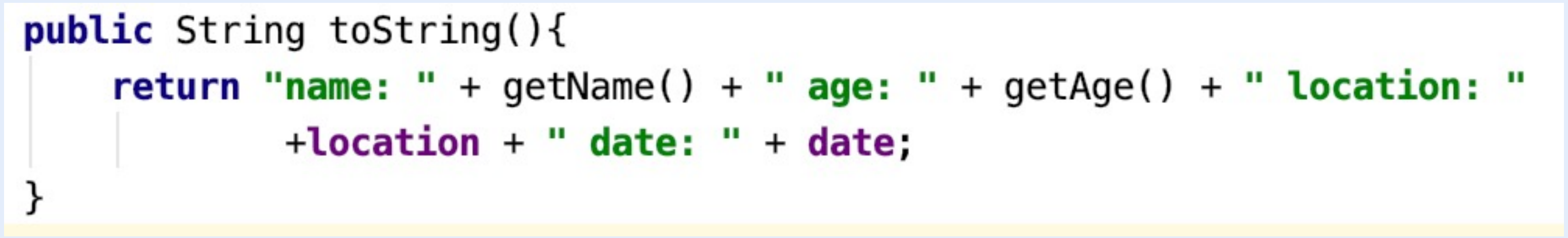
If a class defines a method with same signature as an ancestor, its definition overrides the ancestor's

In Person:



```
public String toString(){
    return "name: " + name + " age: " + age;
}
```

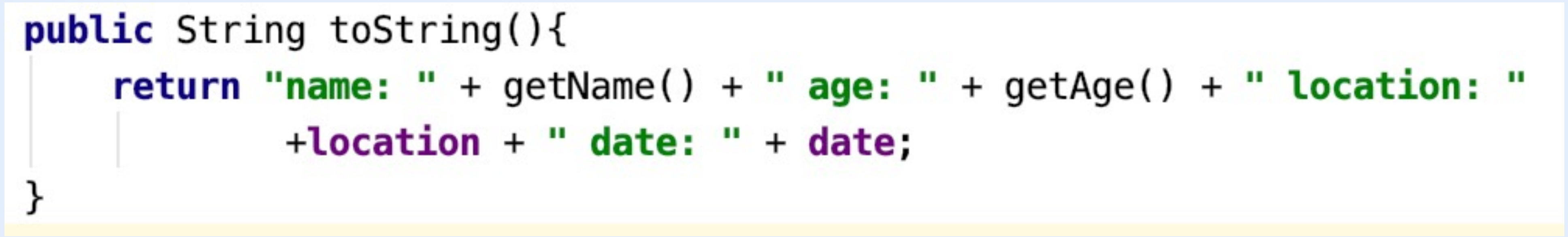
In LostPerson:



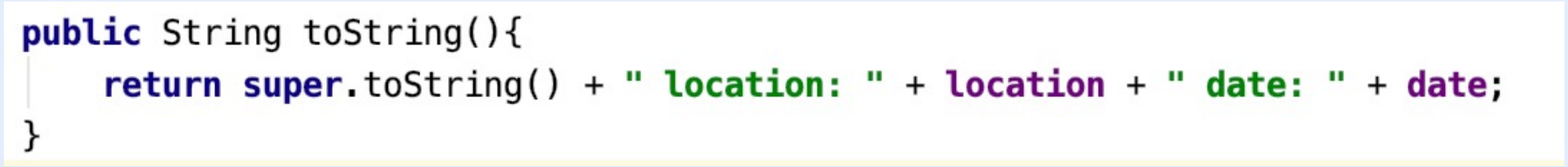
```
public String toString(){
    return "name: " + getName() + " age: " + getAge() + " location: "
            +location + " date: " + date;
}
```

Use overridden methods

inside a method, use super.methodName(args…) to invoke the overridden methods



```java
public String toString(){
    return "name: " + getName() + " age: " + getAge() + " location: "
            +location + " date: " + date;
}
```



```java
public String toString(){
    return super.toString() + " location: " + location + " date: " + date;
}
```

Method Overriding vs Overloading (Polymorphism)

Overriding

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

In Person:

```java
public void greet(String name){
    System.out.println("hello"+ name);
}
```

In LostPerson:

```java
public void greet(String name){
    System.out.println("Find" + name);
}
```

Overloading

two methods have same name but have different signatures

```java
public void greet(String name){
    System.out.println("hello"+ name);
}

public void greet(){
    System.out.println("hello");
}
```

Late Binding

$$\underbrace{\textbf{Person}}\text{ p1 = }\textbf{new }\underbrace{\textbf{LostPerson}}(\ldots)$$

Declared type
(what methods available)

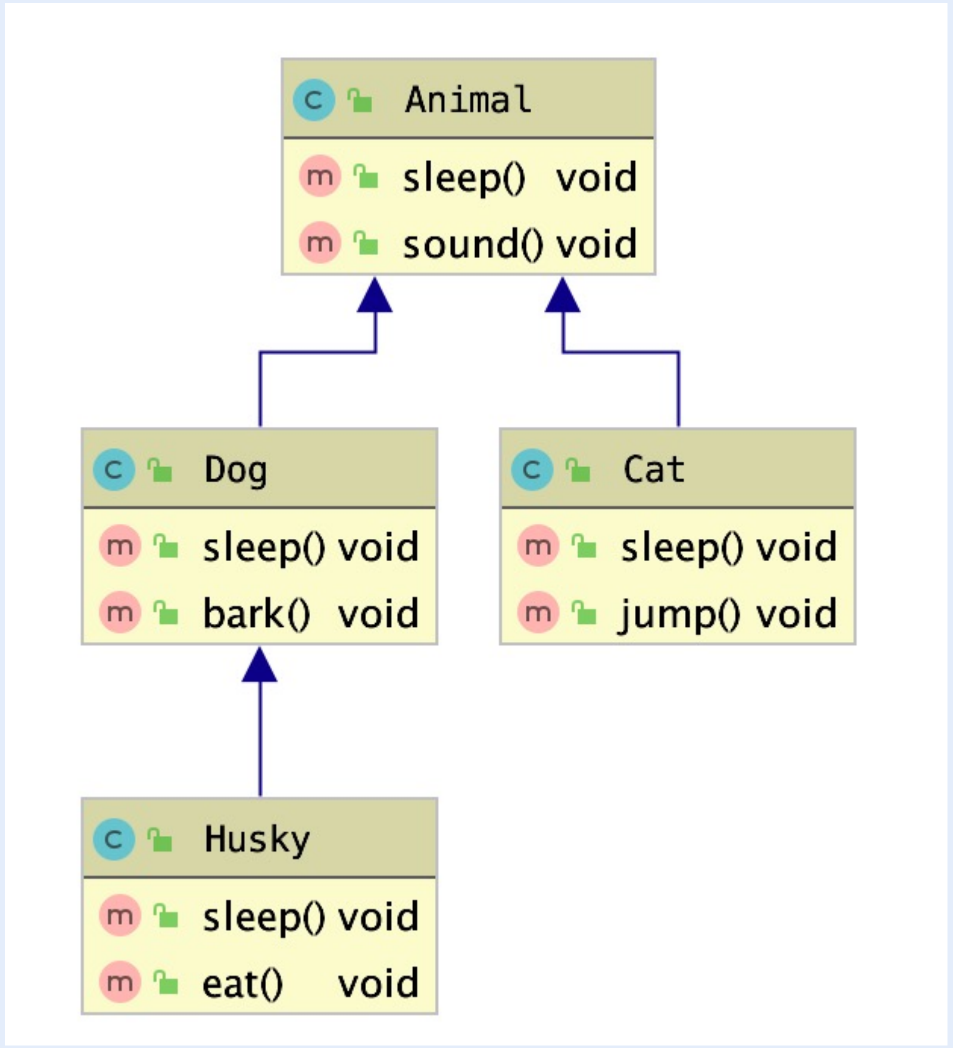
actual type
(which method implementation will be used)

```
Person p1 = new LostPerson(12,"bob","mel",20200502);
System.out.println(p1);
```

which toString method is used??
LostPerson / Person ?

Late Binding

$$\underbrace{\textbf{Person}}_{\text{Declared type (what methods available)}}\ \text{p1} = \textbf{new}\ \underbrace{\textbf{LostPerson}}_{\text{actual type (which method implementation will be used)}}(\ldots)$$



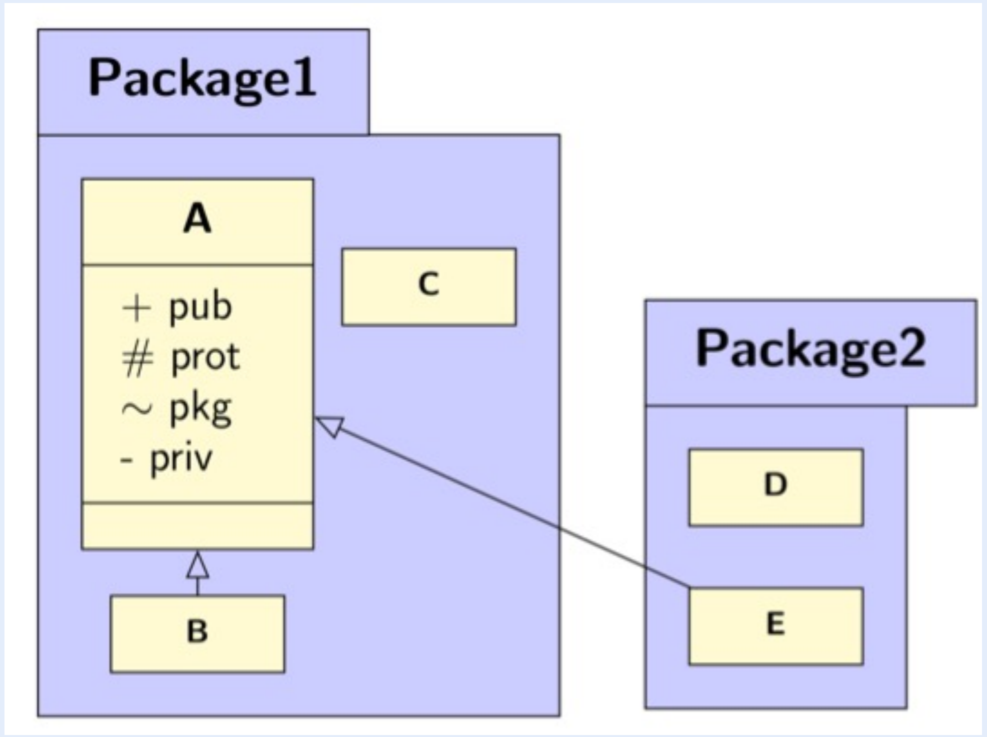
```
Animal a1 = new Dog();
Animal a2 = new Cat();
Dog d1 = new Dog();
Dog d2 = new Husky();

//which statements are wrong/invalid?
```

1. a1.sleep();
2. a1.bark();
3. a2.sleep();
4. a2.sound();
5. d1.bark();
6. d2.eat();

Visibility

private < default(package) < protected < public

(package + subclass)



A sees pub, prot, pkg, priv

B sees pub, prot, pkg

C sees pub, prot, pkg

D sees pub

E sees pub, prot

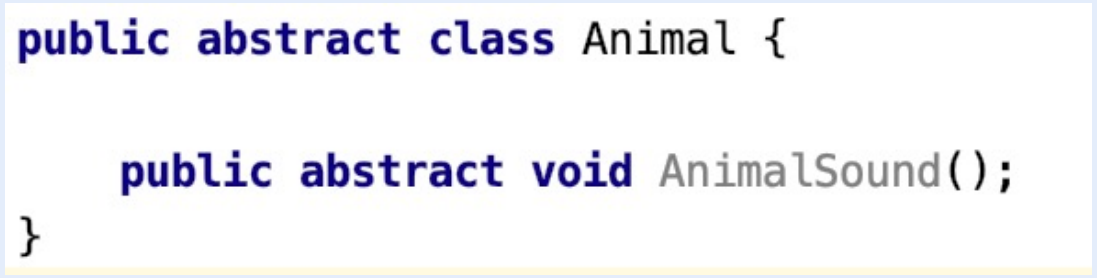
Abstract Method

Form : vis abstract type method(params…) ;

cannot make an instance of a class with abstract method

A class with abstract methods must be declared as abstract

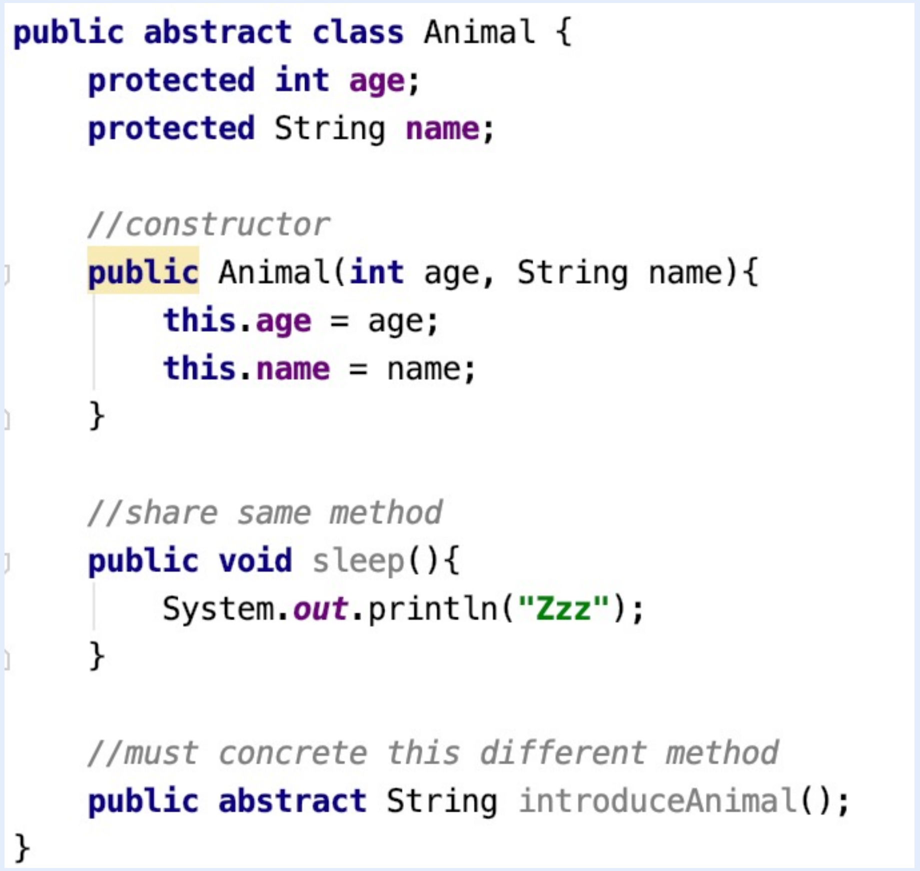
Form: vis abstract class name {….}



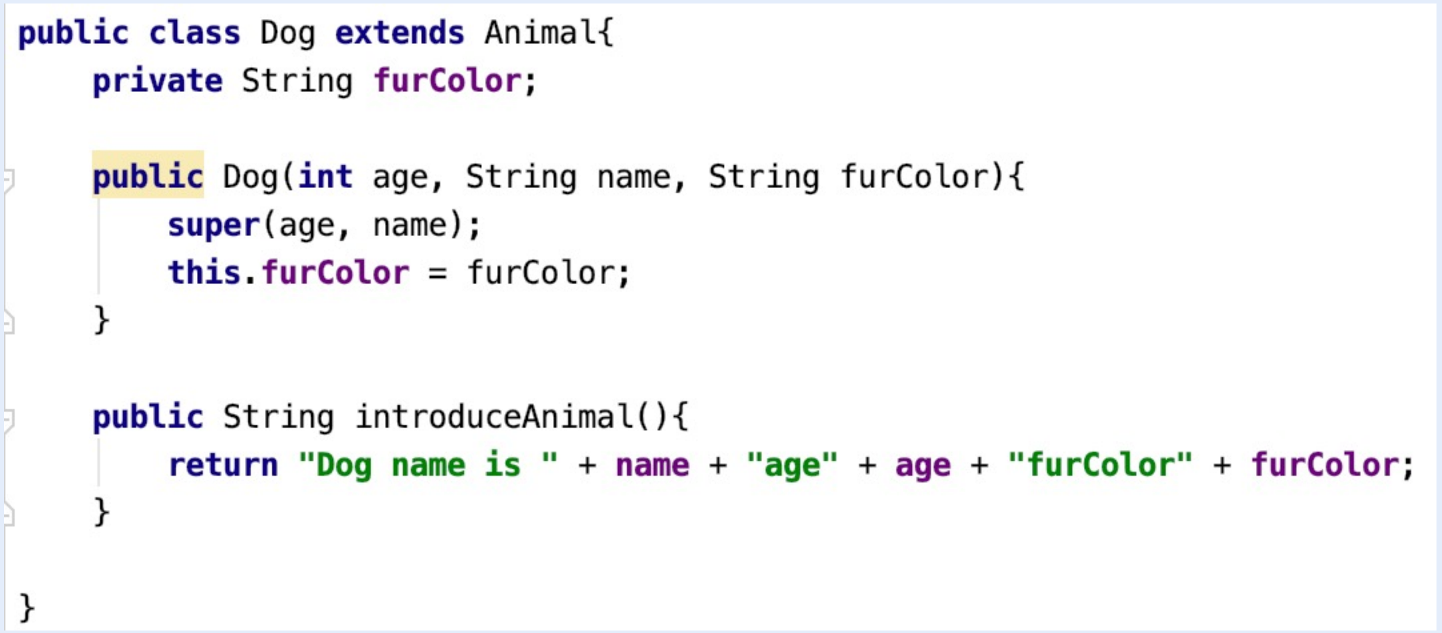
```
public abstract class Animal {

    public abstract void AnimalSound();
}
```

Any concrete class that extends and abstract class must implement(override) all its abstract method!

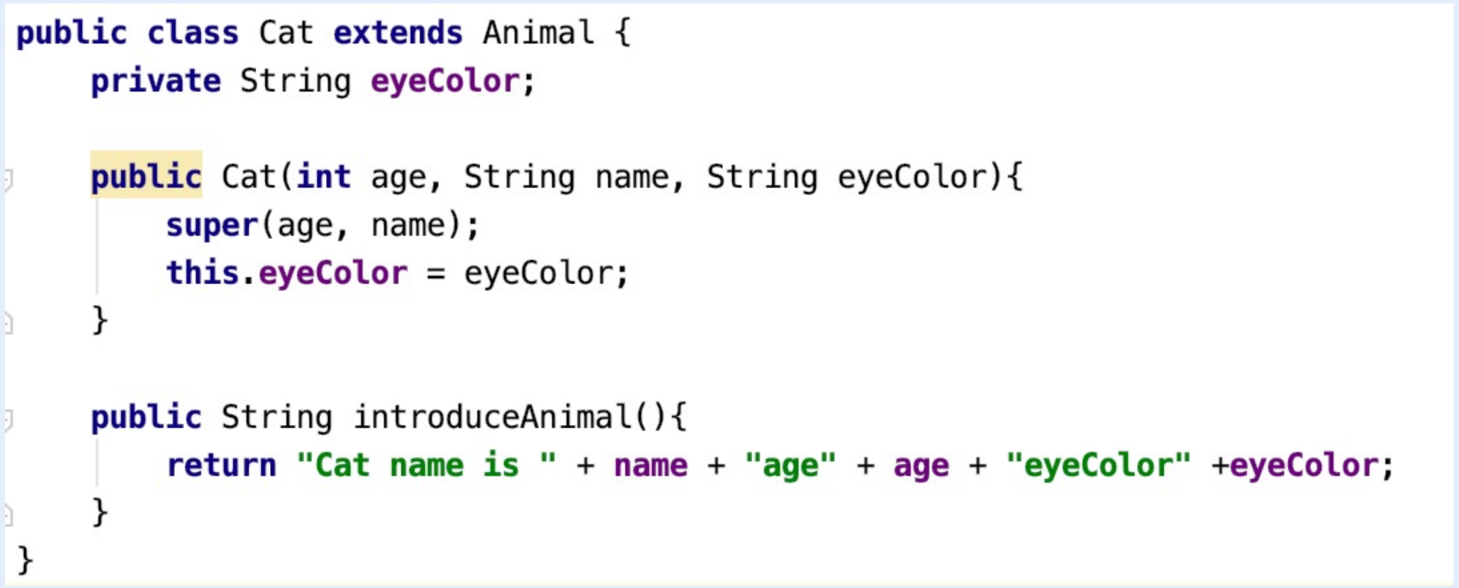
THE UNIVERSITY OF MELBOURNE



```java
public abstract class Animal {
    protected int age;
    protected String name;

    //constructor
    public Animal(int age, String name){
        this.age = age;
        this.name = name;
    }

    //share same method
    public void sleep(){
        System.out.println("Zzz");
    }

    //must concrete this different method
    public abstract String introduceAnimal();
}
```



```java
public class Dog extends Animal{
    private String furColor;

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

    public String introduceAnimal(){
        return "Dog name is " + name + "age" + age + "furColor" + furColor;
    }

}
```



```java
public class Cat extends Animal {
    private String eyeColor;

    public Cat(int age, String name, String eyeColor){
        super(age, name);
        this.eyeColor = eyeColor;
    }

    public String introduceAnimal(){
        return "Cat name is " + name + "age" + age + "eyeColor" +eyeColor;
    }
}
```

Interface

Abstract class allow a number of closely related classes to implement common methods

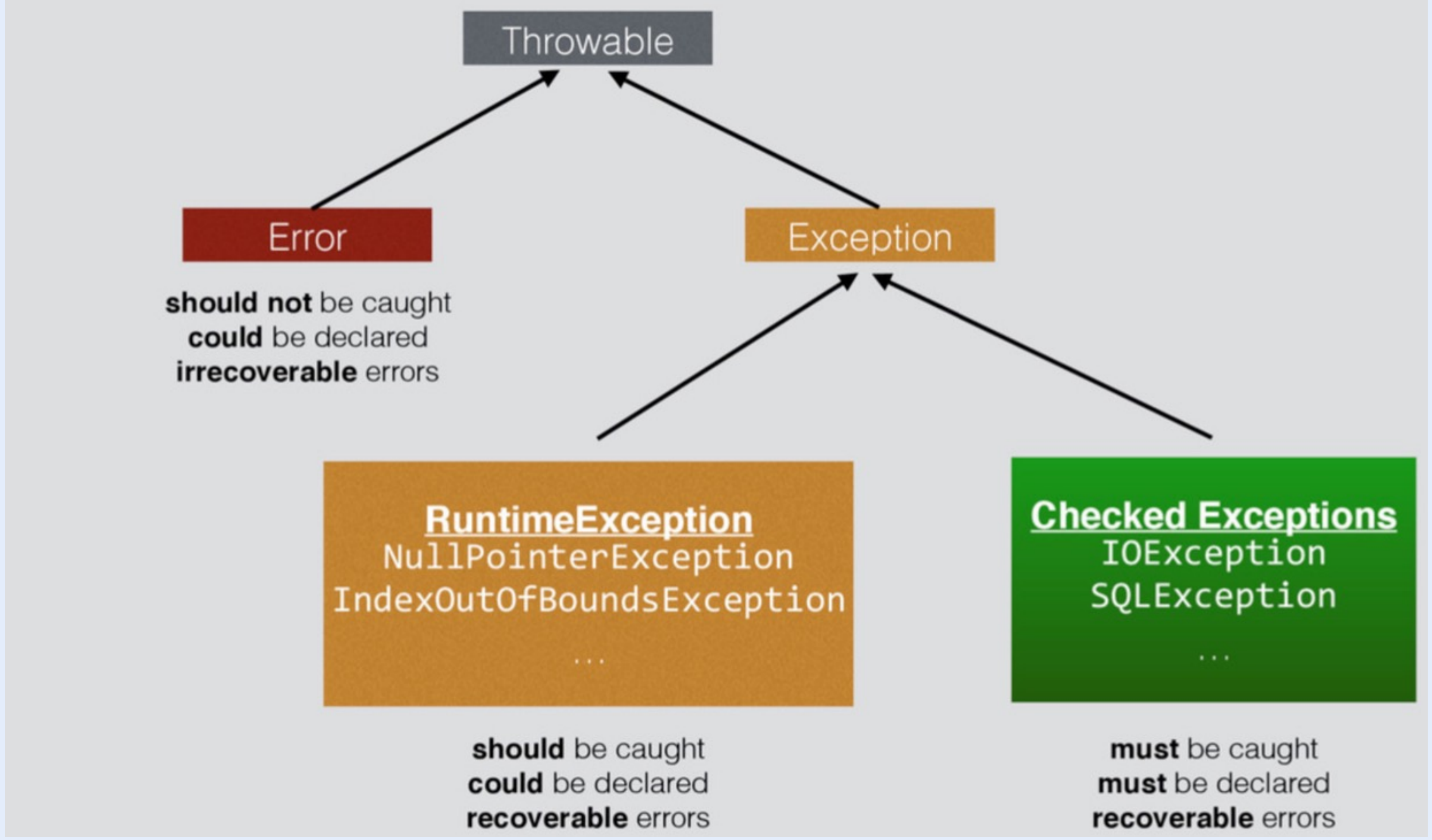
Interface allows unrelated classes to implement common methods

- more abstract than an abstract class
- cannot have instance or class variables
- cannot have non-abstract or static methods

form: public **interface** name{…}

public class name **implements** iface {…}

The Java Exception Hierarchy



Thank you