COMP90041

Programming and Software Development

Semester 2, 2021

Lab 7

Andrew Naughton

andrew.naughton@unimelb.edu.au

Inheritance

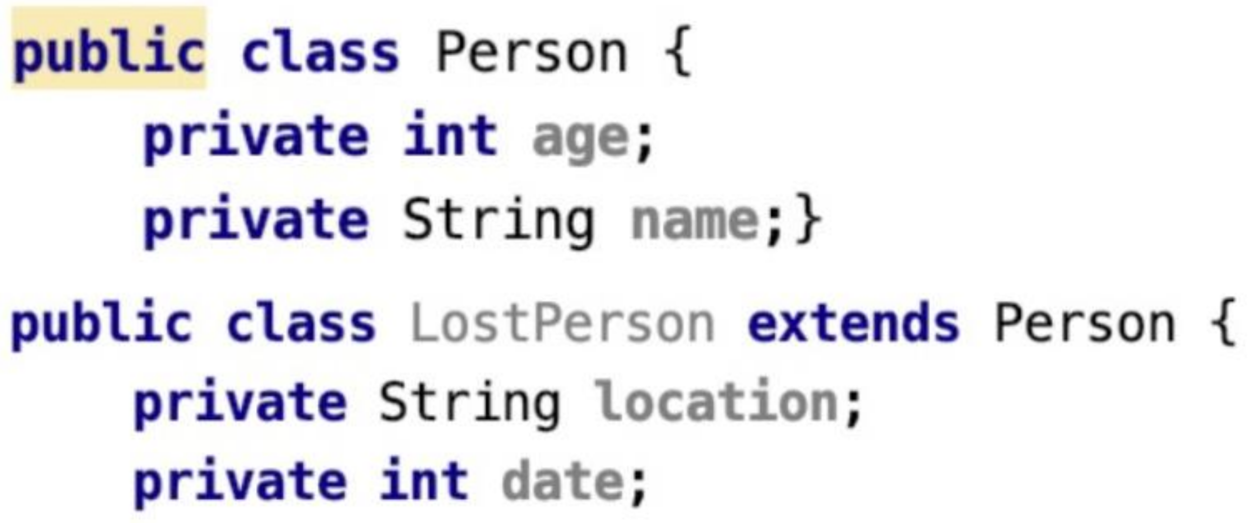
- Defining a class **based on another class**
- Merely need to specify **how it differs from the parent class**
- Keywords: **extends**, inherits, **parent/base/super**, **child/derived/inherited**

Motivation:

1. Many objects in the world are **similar** – why reinvent the wheel?
2. Simplify & improve our code (**less duplication**, better maintainability)

Inheritance

- Objects of **LostPerson** inherit all the instance variables and methods of **Person**…and adds its own!
- No need to re-state inherited instance variables and methods



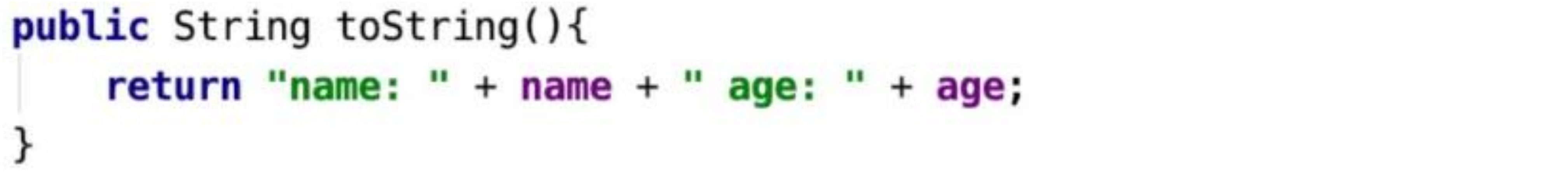
```
public class Person {
    private int age;
    private String name;}

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

- We say that every object of the inherited class is also an object of the base class (i.e. every LostPerson **is a** Person)

Method overriding

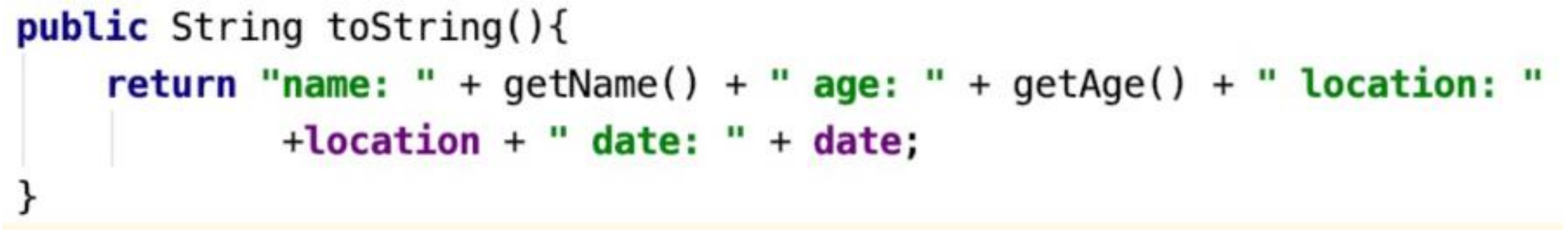
- If a **child class** defines a method with the *same signature* as an **ancestor**, its definition **overrides** the ancestor's

- **Person**



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

- **LostPerson**

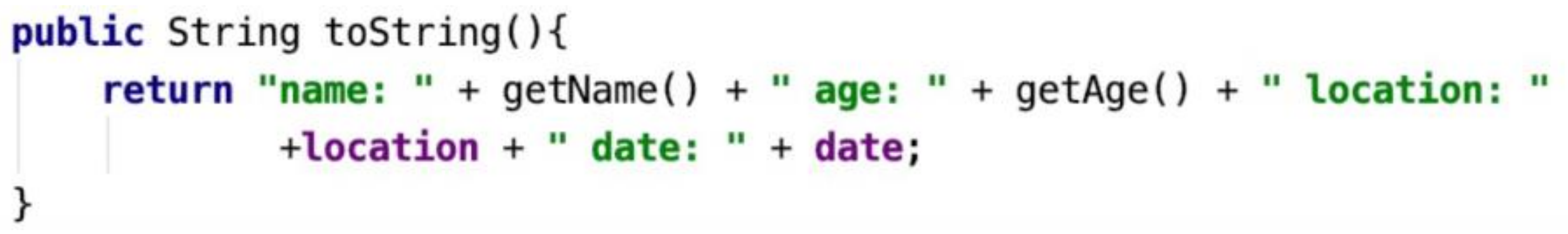


```java
public String toString(){
    return "name: " + getName() + " age: " + getAge() + " location: "
            +location + " date: " + date;
}
```

Method overriding

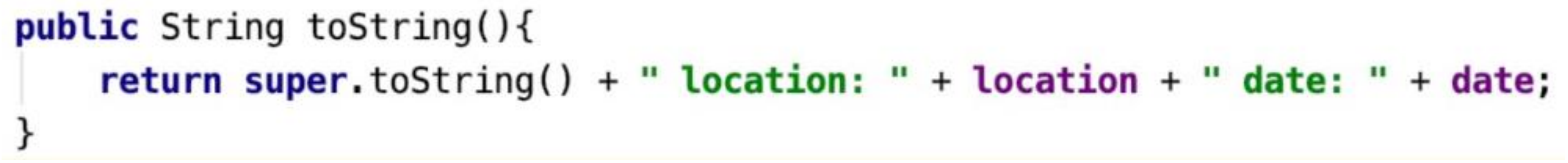
- We can use overridden methods of our parent via: **super.methodName(…)** E.g.:

- Without



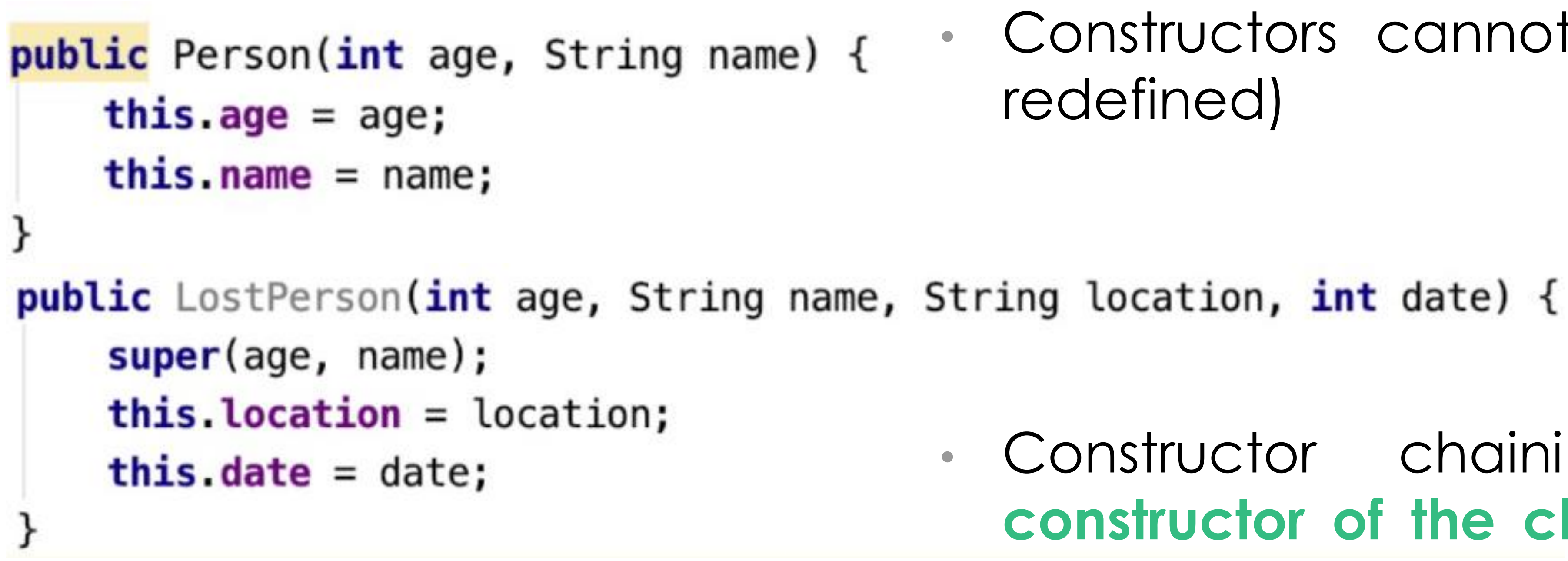
```java
public String toString(){
    return "name: " + getName() + " age: " + getAge() + " location: "
            +location + " date: " + date;
}
```

- With



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

super() Constructor



```
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;
}
```

- Constructors cannot be overridden (i.e. redefined)

- Constructor chaining is when the **constructor of the child** class invokes the **constructor of the parent** class **first**

Overriding vs. Overloading

Overriding

- **Child** can supply its own implementation for a method that also exists in **ancestor**
- **Person**

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

- **LostPerson**

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

Overloading

- Two methods have the **same name** but **different signatures**

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

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

Late binding

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

```
Person person = new LostPerson(60, “Fred”, “Melbourne”, 1234);
```

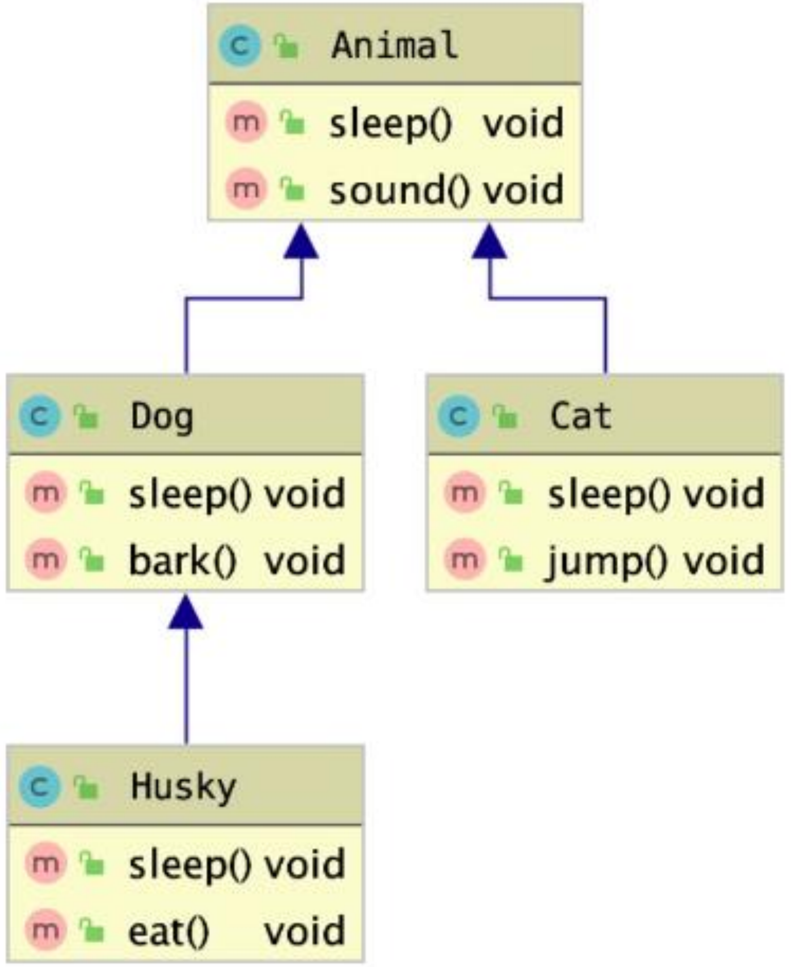
- Whose toString() is called?

Late binding

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

- Which of the following statements are illegal?

```
a1.sleep();        a1.bark();
a2.sleep();        a2.sound();
d1.bark();         d2.eat();
```

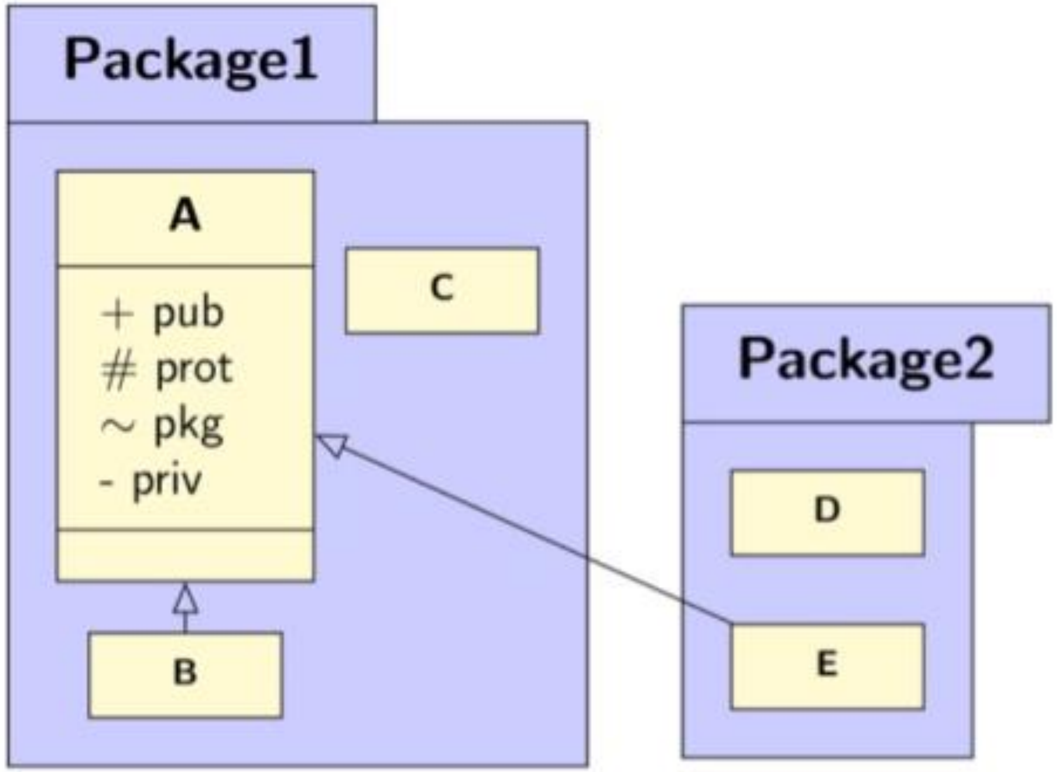


Visibility

private < 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


Package1
A
+ pub
# prot
~ pkg
- priv
C
B
Package2
D
E