2023-02-22

9.1: Inheritance, Superclass, Subclass

OOP & Inheritance

- One of the most useful features of Object-Oriented programming languages (C++, C#, Java, JavaScript, Kotlin, Python, Ruby, Scala, Swift, ActionScript) is **Inheritance**
- **Inheritance** allows your program to efficiently share common code between different objects **(code reuse)**; helps you better organize your program in ways that model the real world; and create smaller units of maintenance and testing.

OOP & Inheritance

- One of the most useful features of Object-Oriented programming languages (C++, C#, Java, JavaScript, Kotlin, Python, Ruby, Scala, Swift, ActionScript) is **Inheritance**
- **Inheritance** allows your program to efficiently share common code between different objects **(code reuse)**; helps you better organize your program in ways that model the real world; and create smaller units of maintenance and testing.

Person
name
address

Student
name
address
locker

Teacher
name
address
office

OOP & Inheritance

- One of the most useful features of Object-Oriented programming languages (C++, C#, Java, JavaScript, Kotlin, Python, Ruby, Scala, Swift, ActionScript) is **Inheritance**
- **Inheritance** allows your program to efficiently share common code between different objects **(code reuse)**; helps you better organize your program in ways that model the real world; and create smaller units of maintenance and testing.

Person
name
address

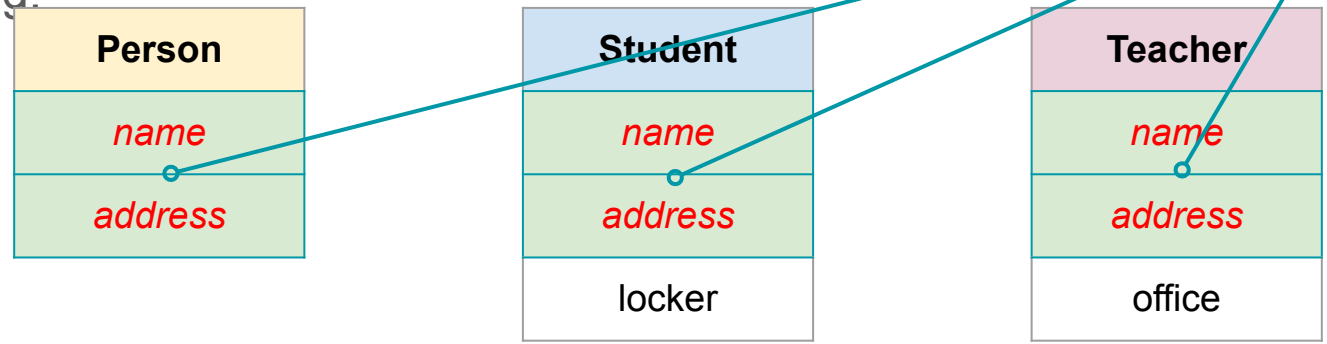
Student
name
address
locker

Teacher
name
address
office

OOP & Inheritance & Generalization

- One of the most useful features of Object-Oriented programm
  (C++, C#, Java, JavaScript, Kotlin, Python, Ruby, Scala, Swif
  Inheritance
- **Inheritance** allows your program to efficiently share common
  different objects **(code reuse)**; helps you better organize you
  ways that model the real world; and create smaller units of maintenance and
  testing.

Identifying and centralizing common information is called "generalization"



OOP & Inheritance & Generalization

- One of the most useful features of Object-Oriented programming languages (C++, C#, Java, JavaScript, Kotlin, Python, Ruby, Scala, Swift, ActionScript) is **Inheritance**
- **Inheritance** allows your program to efficiently share common code between different objects **(code reuse)**; helps you better organize your program in ways that model the real world; and create smaller units of maintenance and testing.

Person
name
address

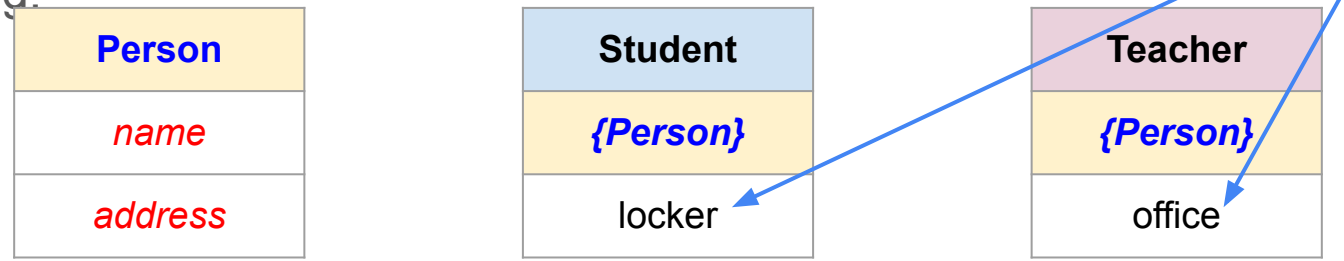
Student
{Person}
locker

Teacher
{Person}
office

OOP & Inheritance & Generalization & Specialization

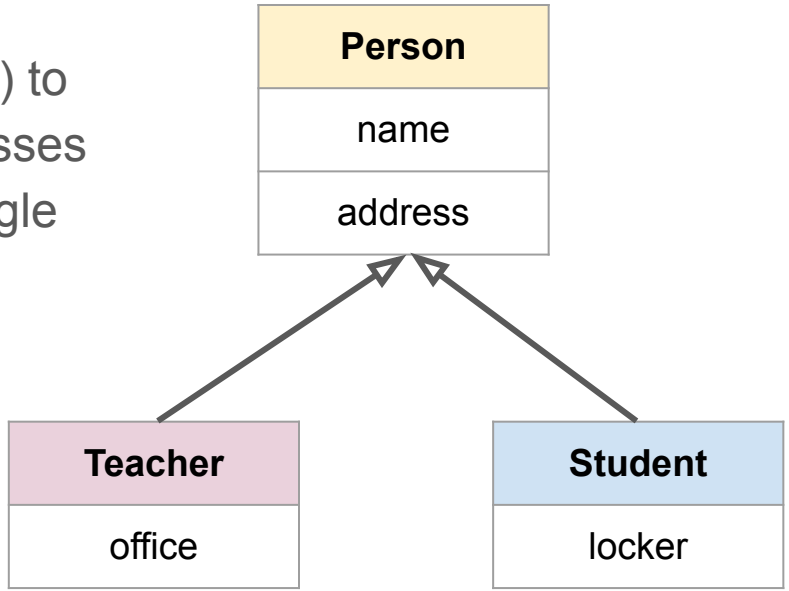
- One of the most useful features of Object-Oriented programming languages (C++, C#, Java, JavaScript, Kotlin, Python, Ruby, Scala, Swift **Inheritance**
- **Inheritance** allows your program to efficiently share common different objects **(code reuse)**; helps you better organize your ways that model the real world; and create smaller units of maintenance and testing.

Placing class-specific information in that class is called "specialization"



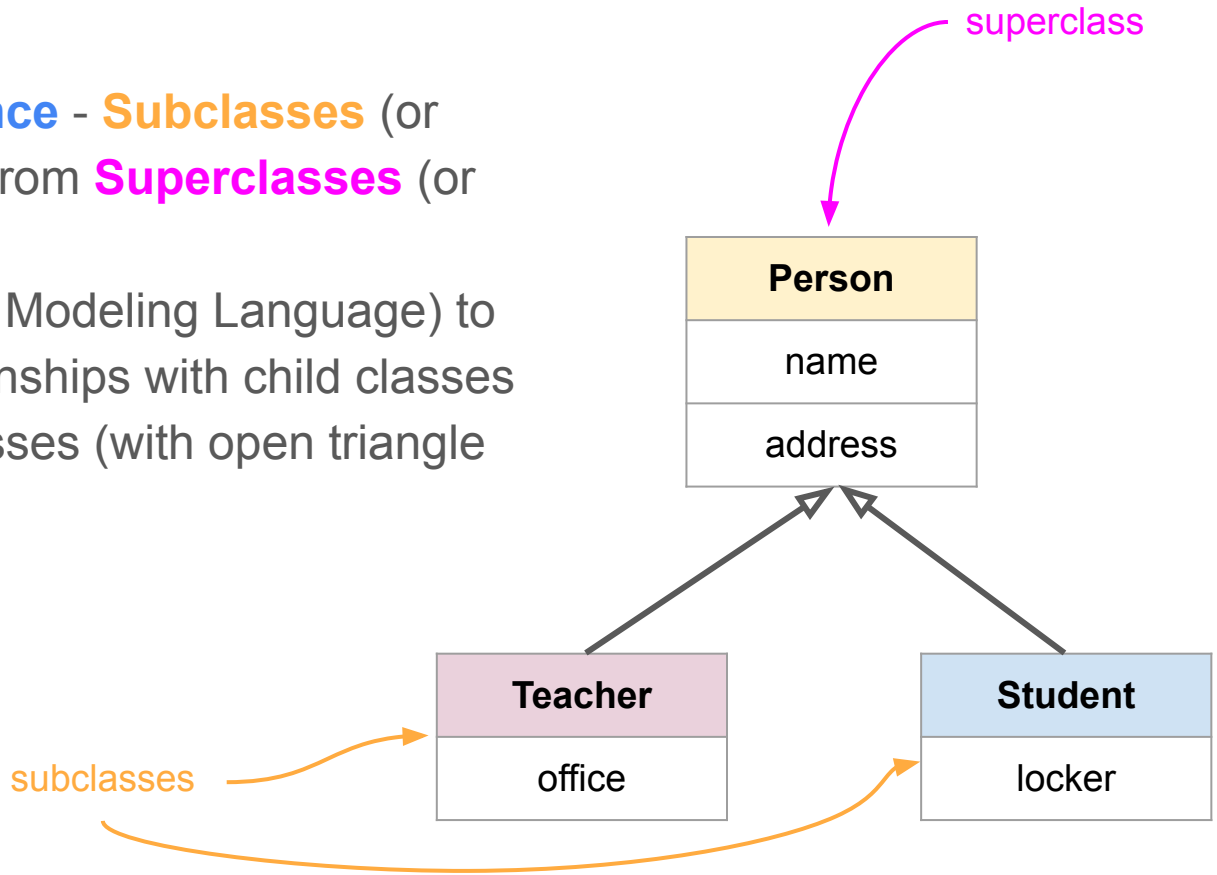
Superclasses & Subclasses & UML

- When using **Inheritance** - **Subclasses** (or child-classes) inherit from **Superclasses** (or parent-classes)
- We use UML (Unified Modeling Language) to describe these relationships with child classes pointing to parent classes (with open triangle endpoints)



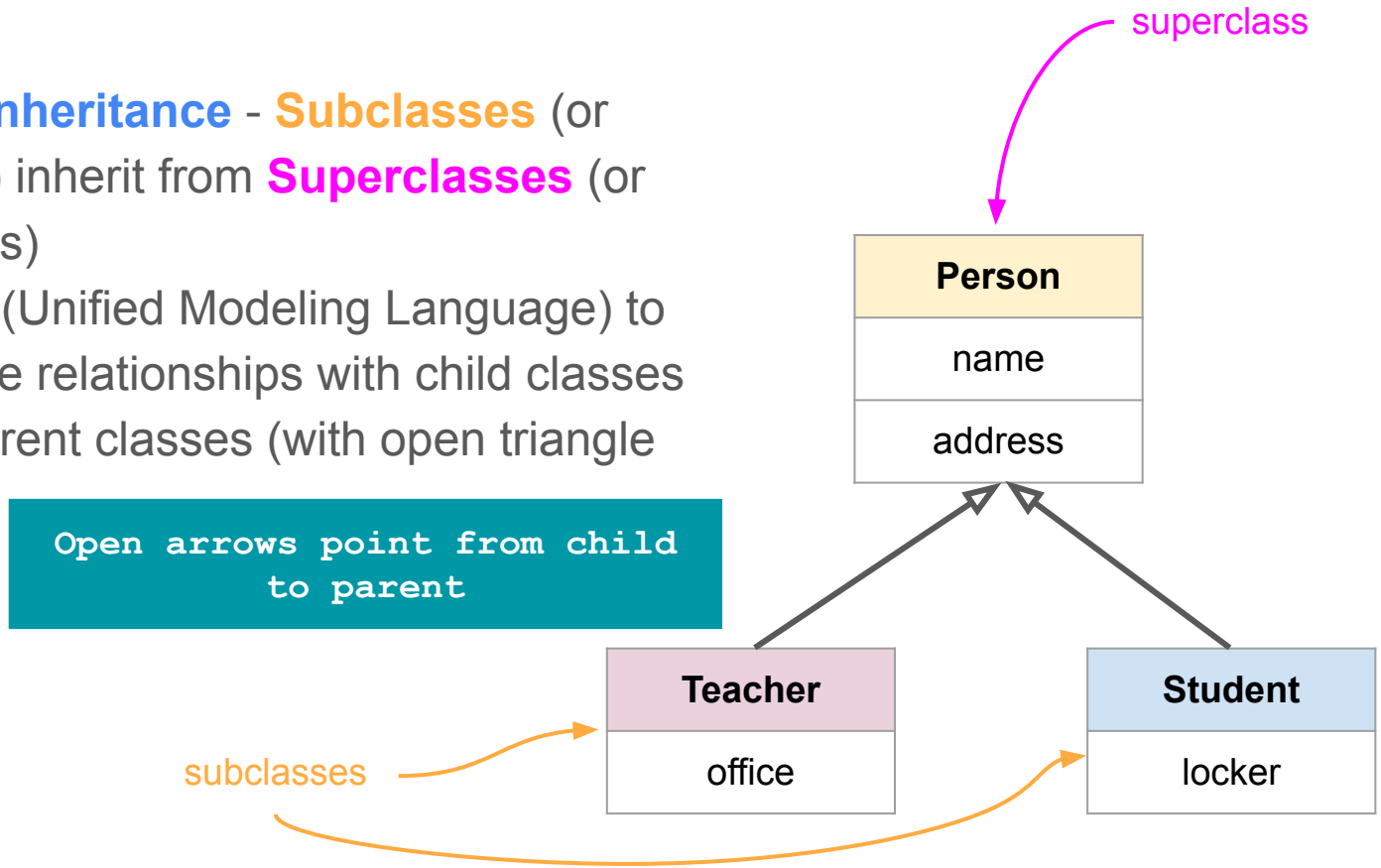
Superclasses & Subclasses & UML

- When using **Inheritance** - **Subclasses** (or child-classes) inherit from **Superclasses** (or parent-classes)
- We use UML (Unified Modeling Language) to describe these relationships with child classes pointing to parent classes (with open triangle endpoints)

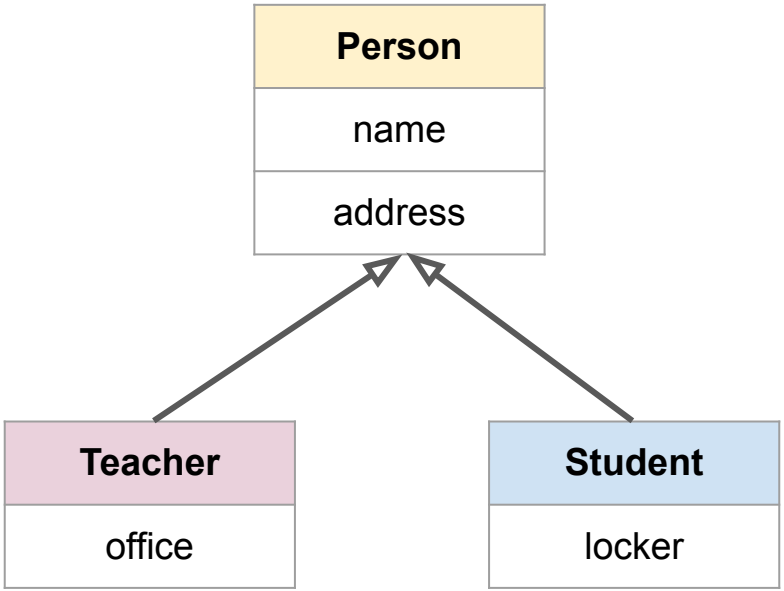


Superclasses & Subclasses & UML

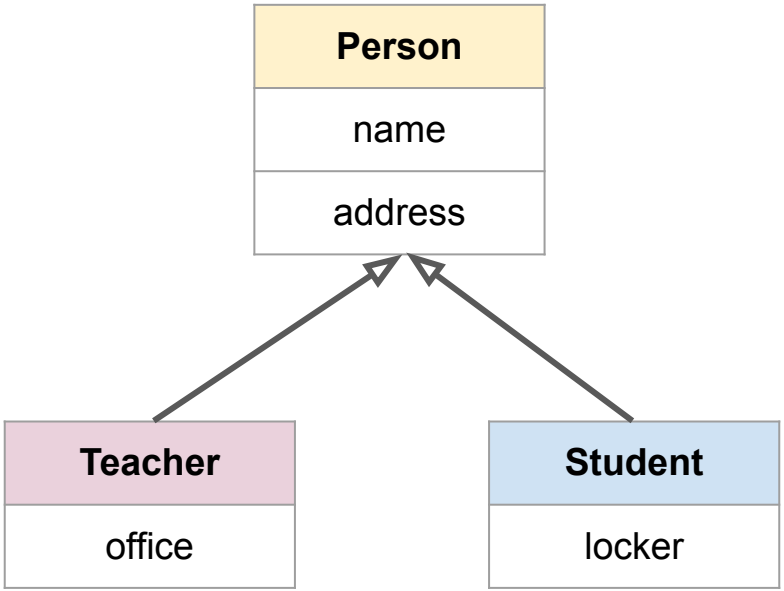
- When using **Inheritance** - **Subclasses** (or child-classes) inherit from **Superclasses** (or parent-classes)
- We use UML (Unified Modeling Language) to describe these relationships with child classes pointing to parent classes (with open triangle endpoints)



Generalization & Specialization

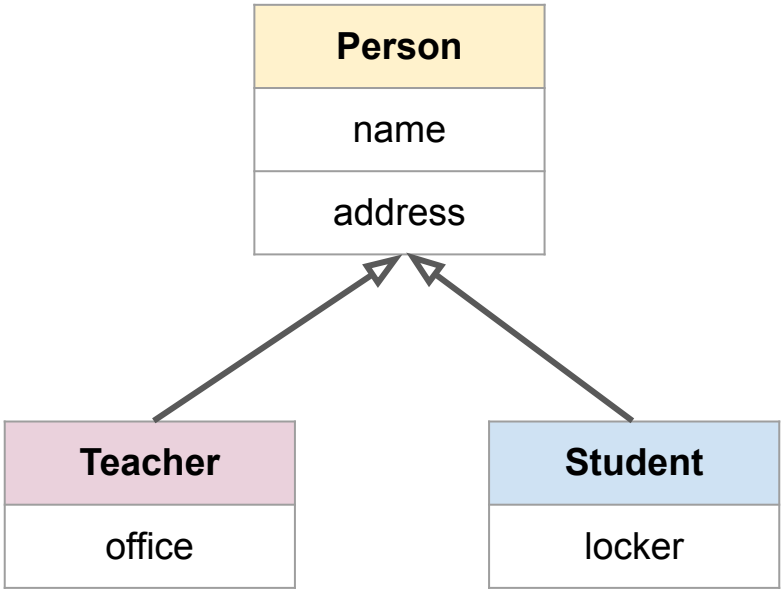


Generalization & Specialization



Q1: What are some additional properties/methods that could be "generalized" to the superclass?

Generalization & Specialization



Q1: What are some additional properties/methods that could be "generalized" to the superclass?

Q2: What are some additional properties/methods that could be "specialized" to the subclasses?

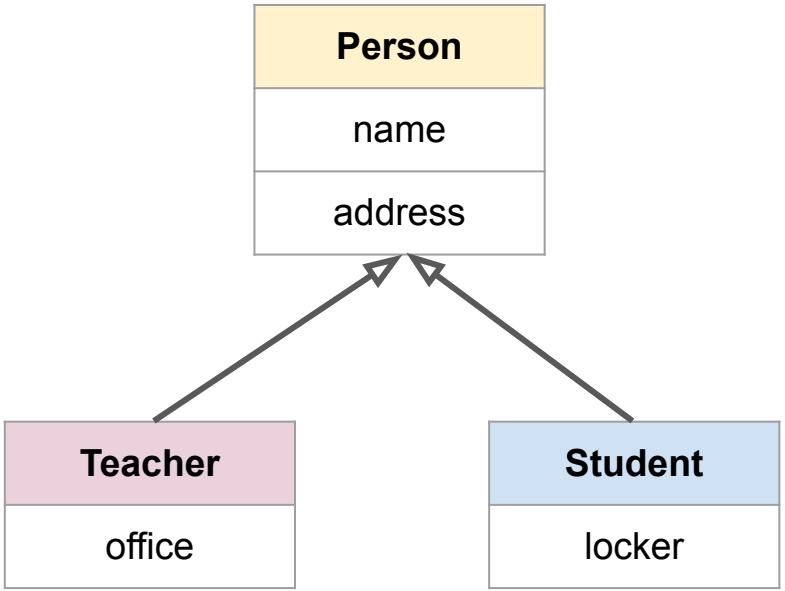
Java & Inheritance

- In Java - any class (**not marked `final`**) can be a superclass - but if a class wants to be a subclass they must use the `extends` keyword

```
class Person {
  public String name;
  public String address;
}

class Teacher extends Person {
  public String office;
}

class Student extends Person {
  public String locker;
}
```



Java & Inheritance

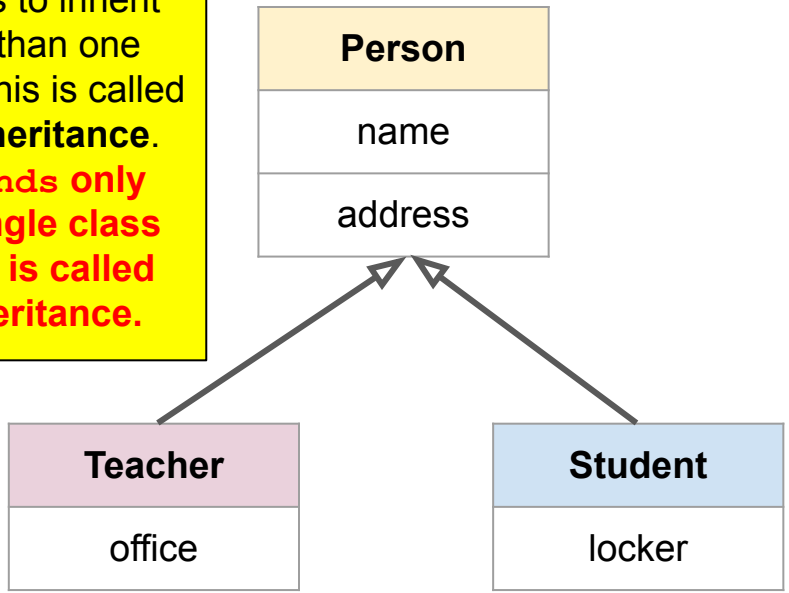
- In Java - any class (**not marke** uperclass - but if a class wants to be a subclass they m keyword

```
class Person {
  public String name;
  public String address;
}

class Teacher extends Person {
  public String office;
}

class Student extends Person {
  public String locker;
}
```

Note: Some OOP languages (C++, Python) allow a class to inherit from more than one superclass - this is called **multiple-inheritance**. **Java `extends` only allows a single class name - this is called single-inheritance.**



Java & Inheritance

Subclasses **inherit** all the variables and methods of their superclass

```
class Person {
  public String name;
  public String address;
  public void printInfo() {
    System.out.println(name + " " + address);
  }
}

class Teacher extends Person {
  public String office;
}

class Student extends Person {
  public String locker;
}
```

Java & Inheritance

Subclasses **inherit** all the variables and methods of their superclass

```
class Person {
  public String name;
  public String address;
  public void printInfo() {
    System.out.println(name + " " + address);
  }
}

class Teacher extends Person {
  public String office;
}

class Student extends Person {
  public String locker;
}
```

```
Person p = new Person();
p.name = "Gary";
p.address = "San Francisco";
p.printInfo();
```

Java & Inheritance

Subclasses **inherit** all the variables and methods of their superclass

```
class Person {
  public String name;
  public String address;
  public void printInfo() {
    System.out.println(name + " " + address);
  }
}

class Teacher extends Person {
  public String office;
}

class Student extends Person {
  public String locker;
}
```

```
Person p = new Person();
p.name = "Gary";
p.address = "San Francisco";
p.printInfo();

Teacher t = new Teacher();
t.name = "Chris";
t.address = "San Mateo";
t.printInfo();
t.office = "215W";
```

Java & Inheritance

Subclasses **inherit** all the variables and methods of their superclass

```
class Person {
  public String name;
  public String address;
  public void printInfo() {
    System.out.println(name + " " + address);
  }
}

class Teacher extends Person {
  public String office;
}

class Student extends Person {
  public String locker;
}
```

```
Person p = new Person();
p.name = "Gary";
p.address = "San Francisco";
p.printInfo();

Teacher t = new Teacher();
t.name = "Chris";
t.address = "San Mateo";
t.printInfo();
t.office = "215W";

Student s = new Student();
s.name = "Beatrice";
s.address = "Colma";
s.printInfo();
s.locker = "B32";
```

Java & Inheritance

Classes that do not use the `extends` keyword automatically extends the `Object` class (has been happening for every class created since Unit 5)

```
class Object {
  public String toString();
  public boolean equals(Object obj);
  ...
}

class Account {
  public String name;
  public double balance;
}
```

Java & Inheritance

Classes that do not use the `extends` keyword automatically extends the `Object` class (has been happening for every class created since Unit 5)

```
class Object {
  public String toString();
  public boolean equals(Object obj);
  ...
}

class Account {
  public String name;
  public double balance;
}
```

```
Object o = new Object();
o.toString();
o.equals(void);
```

Java & Inheritance

Classes that do not use the `extends` keyword automatically extends the `Object` class (has been happening for every class created since Unit 5)

```
class Object {
  public String toString();
  public boolean equals(Object obj);
  ...
}

class Account {
  public String name;
  public double balance;
}
```

```
Object o = new Object();
o.toString();
o.equals(void);


Account a = new Account();
a.toString();
a.equals(void);
a.name = "Amazon";
a.balance = 0.0;
```

Java & Inheritance & `is-a` relationships

Using **Inheritance** results in classes that have `is-a` relationships	
`class Account {}`	

`class Person {}`
`class Teacher extends Person {}`
`class Student extends Person {}`	

`class Animal {}`
`class Dog extends Animal {}`
`class Snake extends Animal {}`	

`class Shape {}`
`class Square extends Shape {}`
`class Circle extends Shape {}`
`class Triangle extends Shape {}`
`class Pentagon extends Shape {}`	

Java & Inheritance & `is-a` relationships

Using **Inheritance** results in classes that have `is-a` relationships	
`class Account {}`	`Account is-a Object`

`class Person {}`
`class Teacher extends Person {}`
`class Student extends Person {}`	

`class Animal {}`
`class Dog extends Animal {}`
`class Snake extends Animal {}`	

`class Shape {}`
`class Square extends Shape {}`
`class Circle extends Shape {}`
`class Triangle extends Shape {}`
`class Pentagon extends Shape {}`	

Java & Inheritance & `is-a` relationships

Using **Inheritance** results in classes that have `is-a` relationships	
`class Account {}`	`Account is-a Object`

`class Person {}`
`class Teacher extends Person {}`
`class Student extends Person {}`
	`Person is-a Object`
	`Teacher is-a Person / Teacher is-a Object`
	`Student is-a Person / Student is-a Object`

`class Animal {}`
`class Dog extends Animal {}`
`class Snake extends Animal {}`	

`class Shape {}`
`class Square extends Shape {}`
`class Circle extends Shape {}`
`class Triangle extends Shape {}`
`class Pentagon extends Shape {}`	

Java & Inheritance & `is-a` relationships

Using **Inheritance** results in classes that have `is-a` relationships	
`class Account {}`	`Account is-a Object`

`class Person {}`
`class Teacher extends Person {}`
`class Student extends Person {}`
	`Person is-a Object`
	`Teacher is-a Person / Teacher is-a Object`
	`Student is-a Person / Student is-a Object`

`class Animal {}`
`class Dog extends Animal {}`
`class Snake extends Animal {}`
	`Animal is-a Object`
	`Dog is-a Animal / Dog is-a Object`
	`Snake is-a Animal / Snake is-a Object`

`class Shape {}`
`class Square extends Shape {}`
`class Circle extends Shape {}`
`class Triangle extends Shape {}`
`class Pentagon extends Shape {}`	

Java & Inheritance & `is-a` relationships

Using **Inheritance** results in classes that have `is-a` relationships	
`class Account {}`	`Account is-a Object`

`class Person {}`
`class Teacher extends Person {}`
`class Student extends Person {}`
	`Person is-a Object`
	`Teacher is-a Person / Teacher is-a Object`
	`Student is-a Person / Student is-a Object`

`class Animal {}`
`class Dog extends Animal {}`
`class Snake extends Animal {}`
	`Animal is-a Object`
	`Dog is-a Animal / Dog is-a Object`
	`Snake is-a Animal / Snake is-a Object`

`class Shape {}`
`class Square extends Shape {}`
`class Circle extends Shape {}`
`class Triangle extends Shape {}`
`class Pentagon extends Shape {}`
	`Shape is-a Object`
	`Square is-a Shape / Square is-a Object`
	`Circle is-a Shape / Circle is-a Object`
	`Triangle is-a Shape / Triangle is-a Object`
	`Pentagon is-a Shape / Pentagon is-a Object`

Java & Inheritance & `is-a` relationships

The **`instanceof`** operator in Java can be used to test for `is-a` relationships

```
class Account {}
```

```
Account a = new Account();
System.out.println(a instanceof Object); // true
System.out.println(a instanceof Account); // true
```

```
class Person {}
class Teacher extends Person {}
```

```
Person p = new Person();
System.out.println(p instanceof Object); // true
System.out.println(p instanceof Person); // true

Teacher t = new Teacher();
System.out.println(t instanceof Object); // true
System.out.println(t instanceof Person); // true
System.out.println(t instanceof Teacher); // true
```

Containment & `has-a` relationships

Another concept utilized by Object-Oriented programming languages is **Containment** - where a class is responsible for maintaining an instance of another class inside itself. This results in a `has-a` relationship. We have been using this quite a lot in our examples and projects

```
class Test {
  public String name;
  public double score;
}

class Course {
  public String name;
  public Test tests[10];
}

class Student {
  public String name;
  public Course courses[5];
}
```

```
Test has-a String (name)

Course has-a String (name)
Course has-a Test[] (tests)

Student has-a String (name)
Student has-a Course[] (courses)
```

Modeling `is-a` & `has-a` Relationships

			`is-a` OR `has-a`
Pet	Cat	Dog	??
Student	Teacher	Class	??
Book	Movie	Media	??
Circle	Shape	Square	??
Lunch	Meal	Food	??

Modeling `is-a` & `has-a` Relationships

			`is-a` OR `has-a`
Pet	Cat	Dog	**Dog is-a Pet** **Cat is-a Pet**
Student	Teacher	Class	**??**
Book	Movie	Media	**??**
Circle	Shape	Square	**??**
Lunch	Meal	Food	**??**

Modeling `is-a` & `has-a` Relationships

			`is-a` OR `has-a`
Pet	Cat	Dog	**Dog is-a Pet** **Cat is-a Pet**
Student	Teacher	Class	**Class has-a Teacher** **Class has-a Student**
Book	Movie	Media	**??**
Circle	Shape	Square	**??**
Lunch	Meal	Food	**??**

Modeling `is-a` & `has-a` Relationships

			`is-a` OR `has-a`
Pet	Cat	Dog	**Dog is-a Pet** **Cat is-a Pet**
Student	Teacher	Class	**Class has-a Teacher** **Class has-a Student**
Book	Movie	Media	**Movie is-a Media** **Book is-a Media**
Circle	Shape	Square	**??**
Lunch	Meal	Food	**??**

Modeling `is-a` & `has-a` Relationships

			`is-a` OR `has-a`
Pet	Cat	Dog	**Dog is-a Pet** **Cat is-a Pet**
Student	Teacher	Class	**Class has-a Teacher** **Class has-a Student**
Book	Movie	Media	**Movie is-a Media** **Book is-a Media**
Circle	Shape	Square	**Circle is-a Shape** **Square is-a Shape**
Lunch	Meal	Food	**??**

Modeling `is-a` & `has-a` Relationships

			`is-a` OR `has-a`
Pet	Cat	Dog	**Dog is-a Pet** **Cat is-a Pet**
Student	Teacher	Class	**Class has-a Teacher** **Class has-a Student**
Book	Movie	Media	**Movie is-a Media** **Book is-a Media**
Circle	Shape	Square	**Circle is-a Shape** **Square is-a Shape**
Lunch	Meal	Food	**Lunch is-a Meal** **Meal has-a Food**

Practice on your own

- CSAwesome 9.1 - Inheritance, Superclass, Subclass
- No Replit today