MET CS 622: Object Oriented Concepts

Reza Rawassizadeh

Why Java?

- The second or third most used programming languages. The first one is JavaScript (very similar notation) and in some rankings the second one is python.

- Most enterprise architectures and software companies are heavily using Java, including Twitter, Facebook, Amazon, LinkedIn, eBay, …

- Many demands for Java in the job market, including Androids (smartphone, wearables, IoT,..) related jobs.

- Knowing OOP is very important. Still there are lots of market opportunities for OOP software, e.g. game industry (largest entertainment market in the world).

What you need to do?

- Java Standard Edition Development Kit v.8 (JDK 8).

  https://www.oracle.com/technetwork/java/javase/downloads/index.html

- Install a decent IDE, e.g. Eclipse, which is the most popular IDE for Java Development.

  https://www.eclipse.org/downloads/packages

- Setting your OS environment PATH to point to the latest JDK you have installed.

```
java -version
```

- Setting your OS environment CLASSPATH to the Java “bin” folder. Otherwise, you might get following error:

```
Exception in thread “main”
java.lang.NoClassDefFoundError: <Your class name>
```

- Setting your OS environment JAVA_HOME to the JDK path.

What you need to do?

- Java Standard Edition Development Kit v.12 (JDK 12).

  https://www.oracle.com/technetwork/java/javase/downloads/index.html

- Install a decent IDE, e.g. for Java Development.

  https://www.eclipse.org

- Setting your OS environm u have installed.

```
java -versic
```

- Setting your OS environm er. Otherwise, you might get following error:

```
Exception in thread “main”
java.lang.NoClassDefFoundError: <Your class name>
```

- Setting your OS environment JAVA_HOME to the JDK path.

Be patient while you are configuring your OS for the Java Development. Like other installation it might be a cumbersome task.

Outline

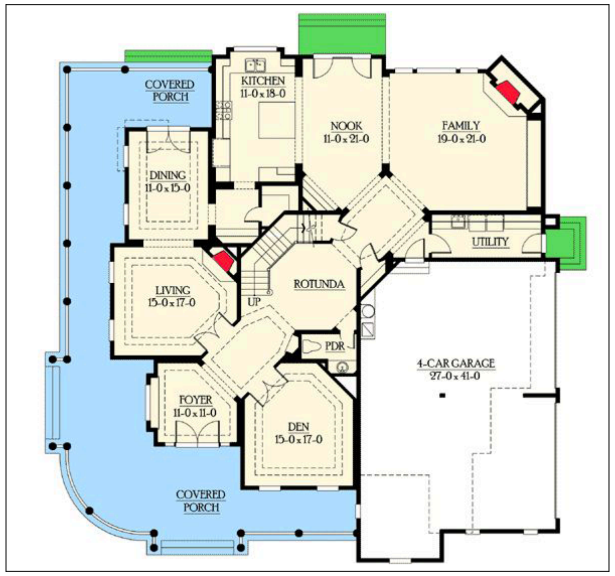
- Class and Objects
- Encapsulation and Access Control
- Inheritance
- Polymorphism
- Abstraction

Outline

- **Class and Objects**
- Encapsulation and Access Control
- Inheritance
- Polymorphism
- Abstraction

What do we need when we plan to create something complicated?

Class and Objects

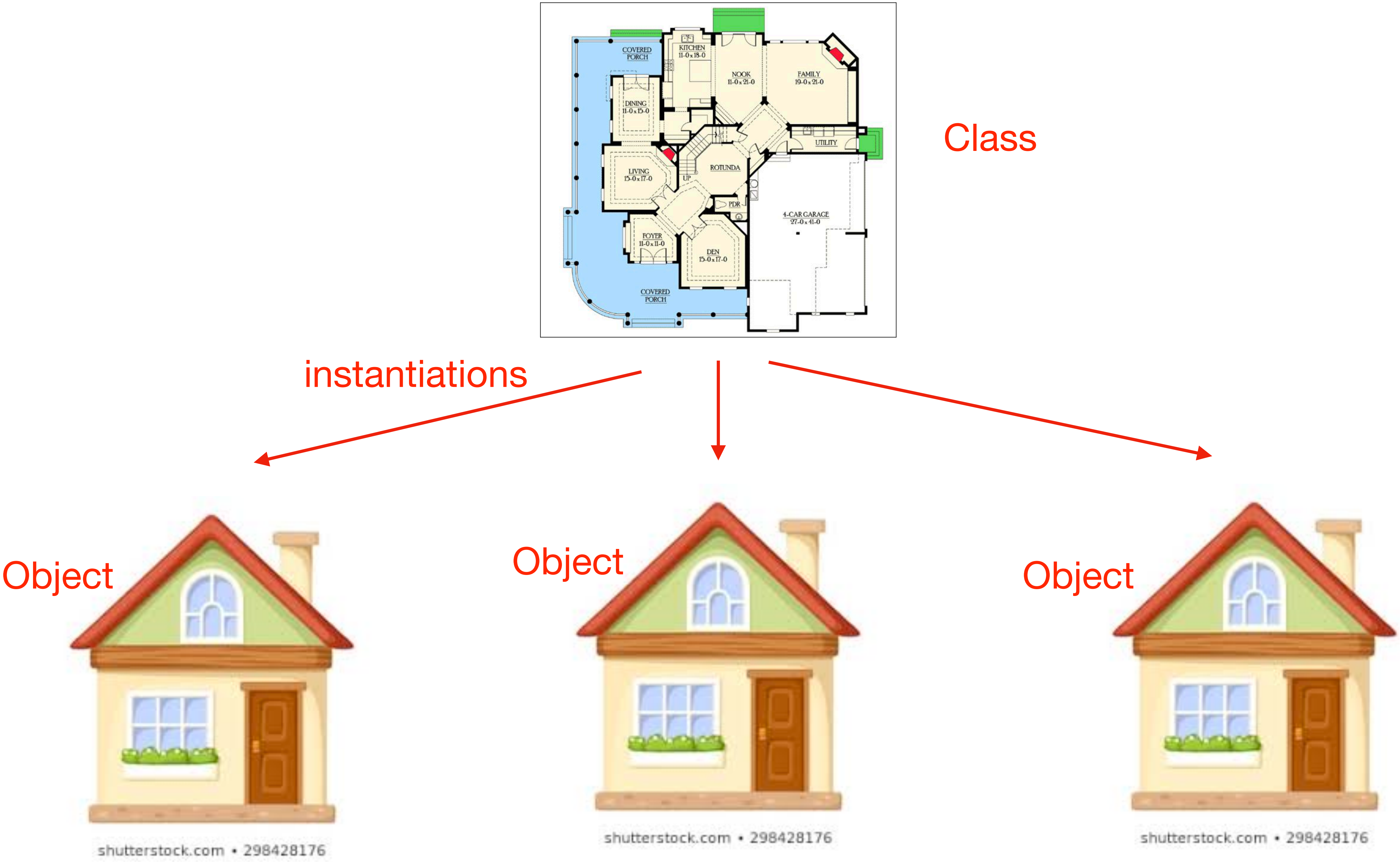


shutterstock.com • 298428176

shutterstock.com • 298428176

shutterstock.com • 298428176

Class and Objects



Class

instantiations

Object

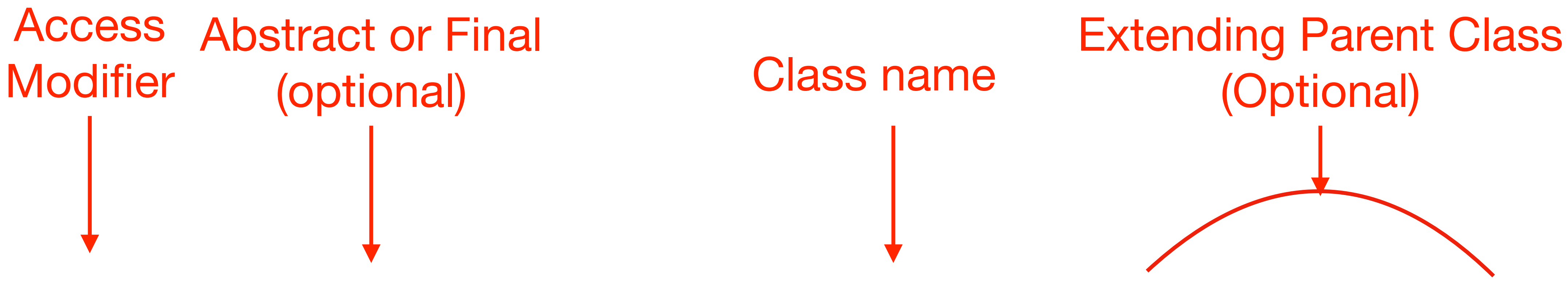
Object

Object

Class Definitions

```
public          class                   Animal {

 …methods and fields will be defined here

}
```

Class Definitions



Access Modifier

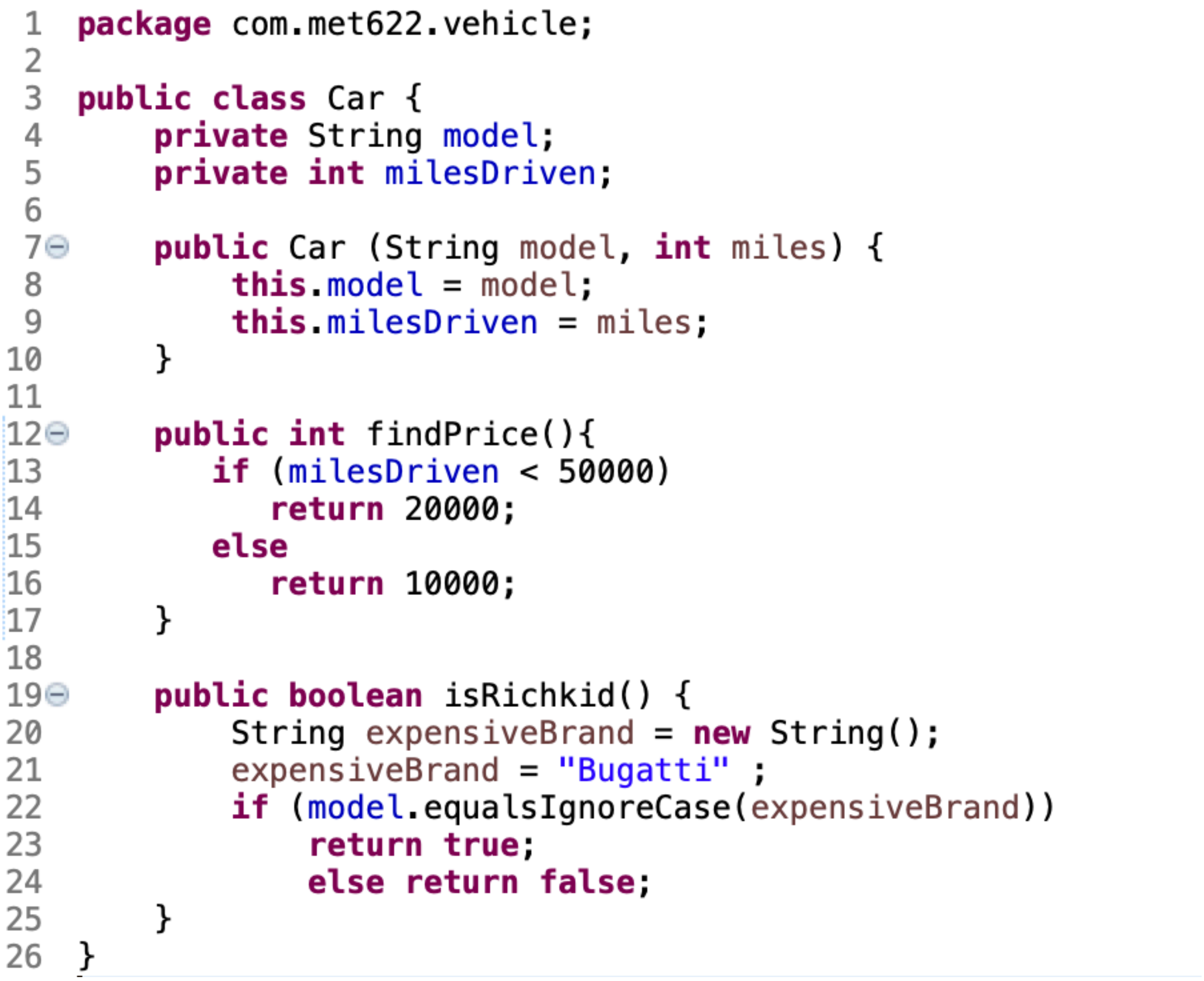
Abstract or Final (optional)

Class name

Extending Parent Class (Optional)

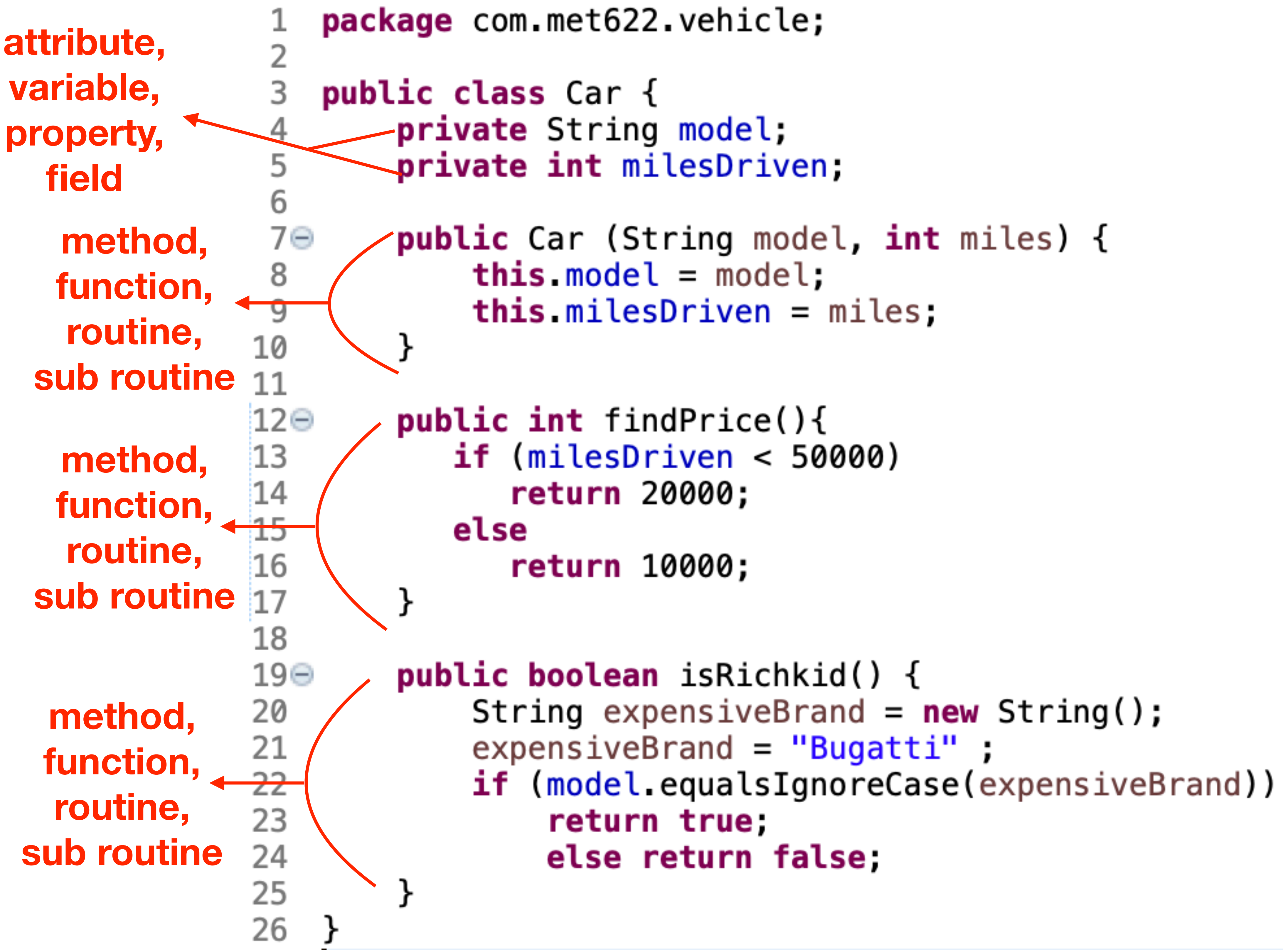
```
public abstract class Elephant extends Animal {

 …methods and fields will be defined here

}
```

Classes and Objects



```java
package com.met622.vehicle;

public class Car {
    private String model;
    private int milesDriven;

    public Car (String model, int miles) {
        this.model = model;
        this.milesDriven = miles;
    }

    public int findPrice(){
       if (milesDriven < 50000)
          return 20000;
       else
          return 10000;
    }

    public boolean isRichkid() {
        String expensiveBrand = new String();
        expensiveBrand = "Bugatti" ;
        if (model.equalsIgnoreCase(expensiveBrand))
            return true;
            else return false;
    }
}
```

Classes and Objects



```java
package com.met622.vehicle;

public class Car {
    private String model;
    private int milesDriven;

    public Car (String model, int miles) {
        this.model = model;
        this.milesDriven = miles;
    }

    public int findPrice(){
       if (milesDriven < 50000)
          return 20000;
       else
          return 10000;
    }

    public boolean isRichkid() {
        String expensiveBrand = new String();
        expensiveBrand = "Bugatti" ;
        if (model.equalsIgnoreCase(expensiveBrand))
            return true;
            else return false;
    }
}
```

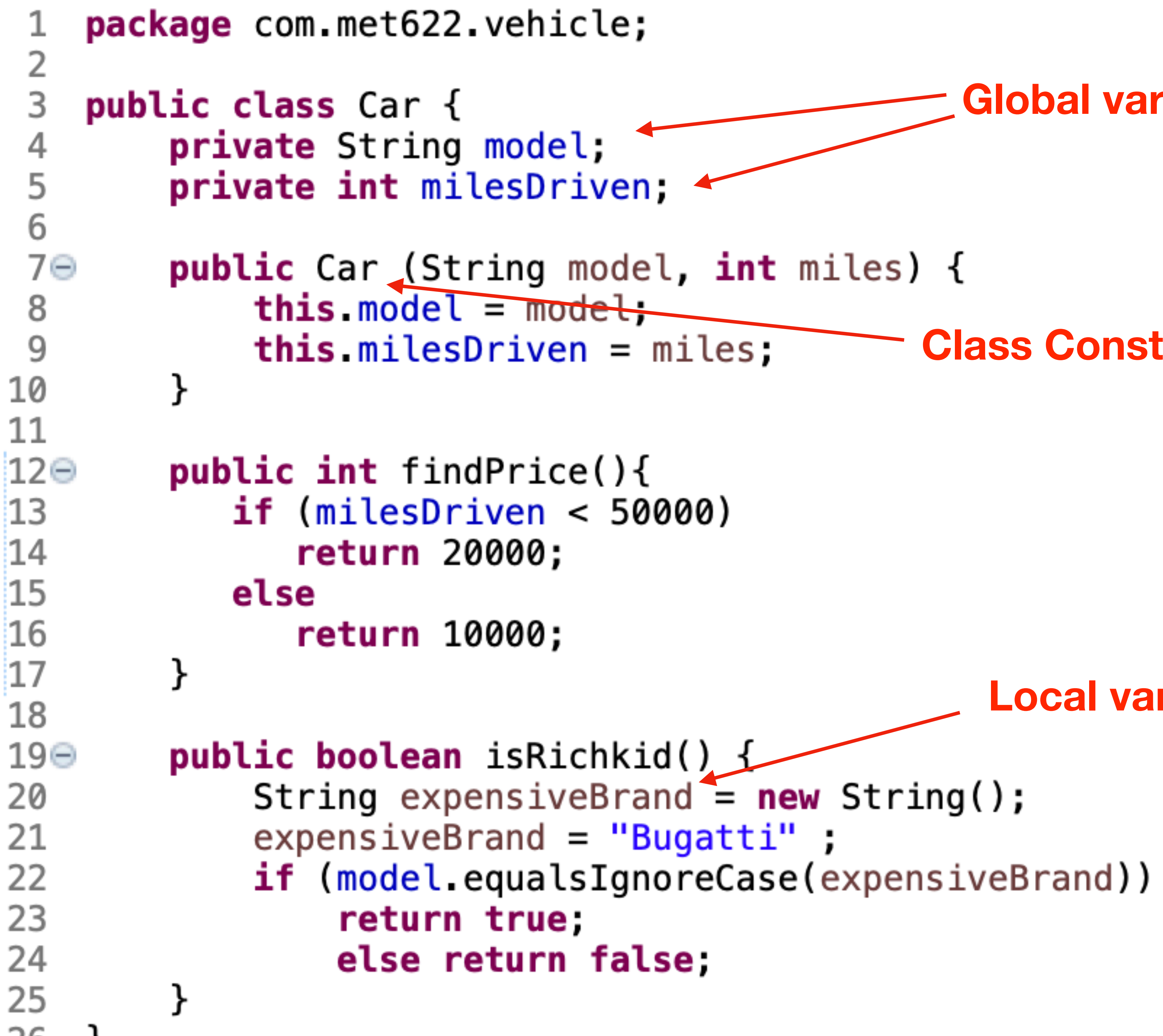
attribute,
variable,
property,
field

method,
function,
routine,
sub routine

method,
function,
routine,
sub routine

method,
function,
routine,
sub routine

Classes and Objects



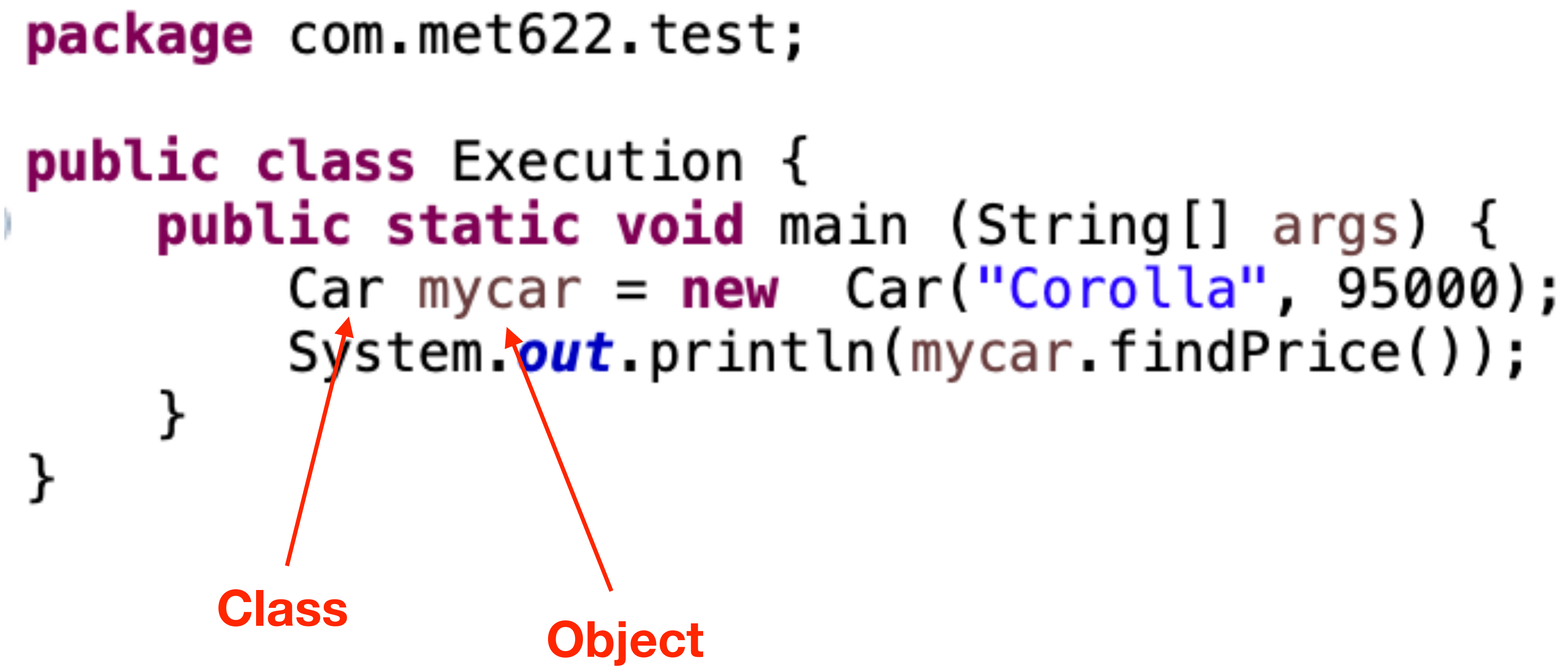
```java
package com.met622.vehicle;

public class Car {
    private String model;
    private int milesDriven;

    public Car (String model, int miles) {
        this.model = model;
        this.milesDriven = miles;
    }

    public int findPrice(){
       if (milesDriven < 50000)
          return 20000;
       else
          return 10000;
    }

    public boolean isRichkid() {
        String expensiveBrand = new String();
        expensiveBrand = "Bugatti" ;
        if (model.equalsIgnoreCase(expensiveBrand))
            return true;
            else return false;
    }
}
```

Global variables

Class Constructor

Local variable

Classes and Objects



```java
package com.met622.test;

public class Execution {
    public static void main (String[] args) {
        Car mycar = new  Car("Corolla", 95000);
        System.out.println(mycar.findPrice());
    }
}
```

Constructor

- Every class has at least one constructor. In the case that no constructor is declared, the compiler will automatically insert a default no-argument constructor.
- The first statement of every constructor is either a call to another constructor within the class, using `this()`, or a call to a constructor in the parent class, using `super()`.
- Java compiler automatically inserts a call to the no-argument constructor `super().` It means we can skip writing the `super()`keyword explicitly inside the constructor, the java compiler insert it on its own..

Constructor

- Every class has at least one constructor. In the case that
  no constr automatically
  insert a d
- The first s ther a call to
  another c his(), or a
  call to a c g super().
- Java co ll to the no-
  argument s we can skip
  writing inside the
  constructor, the Java compiler insert it on its own..

`super()`, is a statement that explicitly calls a **parent constructor** and may only be used in the first line of a constructor of a child class.

`super()`, keyword references a member defined in a parent class and may be used throughout the child class.

Static

- Static makes the variable/method shared among all instances (objects) of a class and we can use class name to refer to it.

- It means only one single instance of static variable/method exists in JVM. Even, if we create many instances of that class, there is only one static variable assigned in the memory.

- Assume we have following code.

```
Public Class Teststatic {
   String a = new String()
   Static String b = new String()
```

If we make 10 instances from `Teststatic` class, every `‘a’` variable occupies a space in the memory, this means we will have 10 `‘a’` in the memory. However, we will have only one space assigned to `‘b’` in the memory.

Naming Conventions in Java

- Class name must start with **capital** letter.

- Object name must start from **lowercase.**

- Parameter names must start with a **small** letter. Except they are **final static constant** variable. In that case everything should be written in capital. e.g.

```
public static final String SERVER_ADDRESS=“192.168.1.1”;
```

- Package names are **all lowercase**, separated with dots and should have a meaning, e.g. `edu.bu.met622.test`

Outline

- Class and Objects
- **Encapsulation and Access Control**
- Inheritance
- Polymorphism
- Abstraction

Access Names

- **Public** variables or methods are visible to all classes everywhere.

- **Private** variables or methods are accessible only to methods inside the same class. They are not visible outside the class. Therefore, overriding private variable isn’t possible.

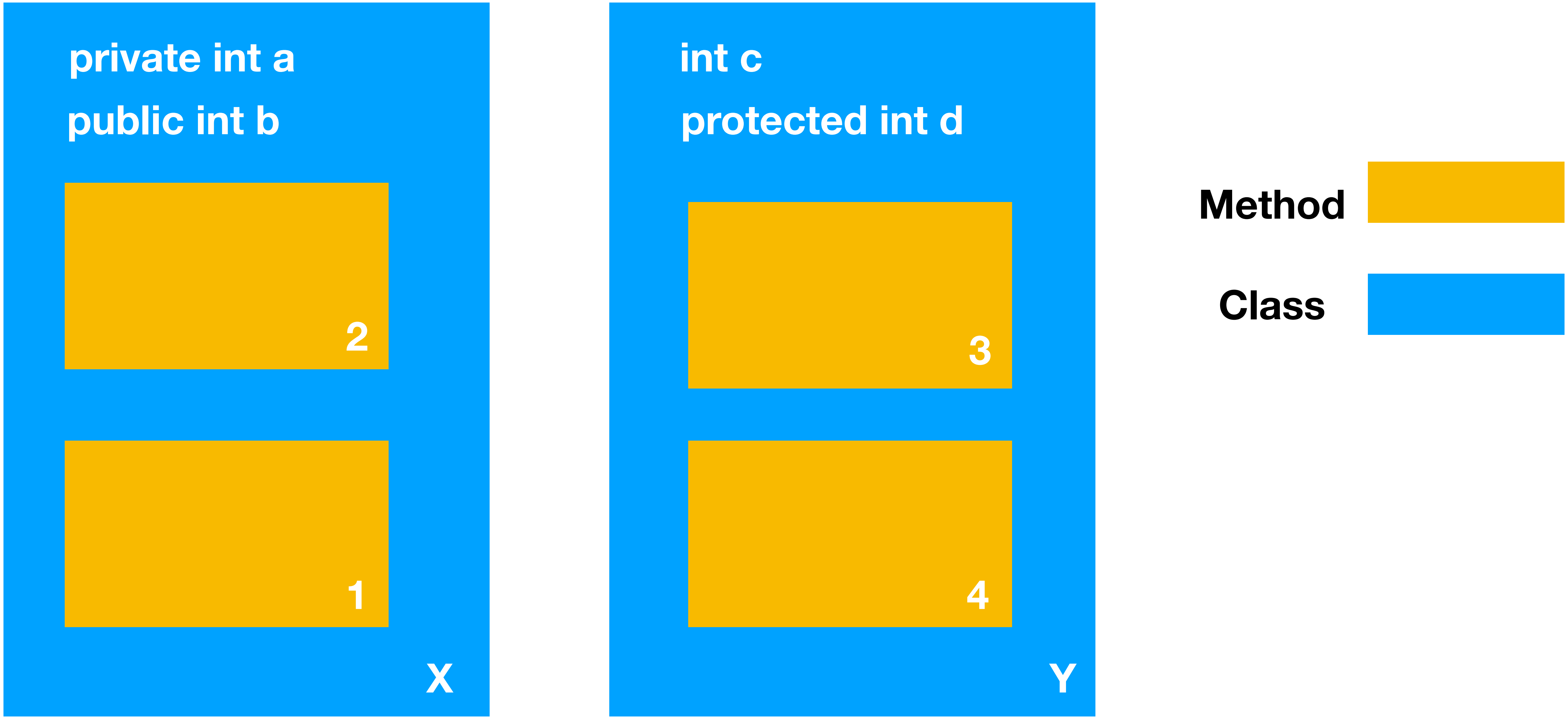
- **Protected** variables or methods can be seen by subclasses or packages member. We use `protected` keyword

- **Package Private** can be only seen inside the package in which it was declared.

Access Controls Details

	Private	Default (Package Access)	Protected	Public
Member in the Class	Yes	Yes	Yes	Yes
Member in another Class, but same package	No	Yes	Yes	Yes
Member in Superclass in different package	No	No	Yes	Yes
Member in a class, but in a different package	No	No	No	Yes

Lets do some example



private int a

public int b

2

1

X

int c

protected int d

3

4

Y

Method 🟨

Class 🟦

Encapsulation

- Usually all class parameters will be defined as private and we use getter/setter methods to access them.

- Getter/setters hides the unsafe access to the class parameters/fields.

correct, but not recommended:

```
public class TestEncapsulation {
       public int param;
```

correct:

```
public class TestEncapsulation {
       private int param;
       public int getParam() {
          return param;
       }
       public void setParam(inP) {
           this.param = inP;
       }
```

- If the data type is boolean instead of get.. we will use is…

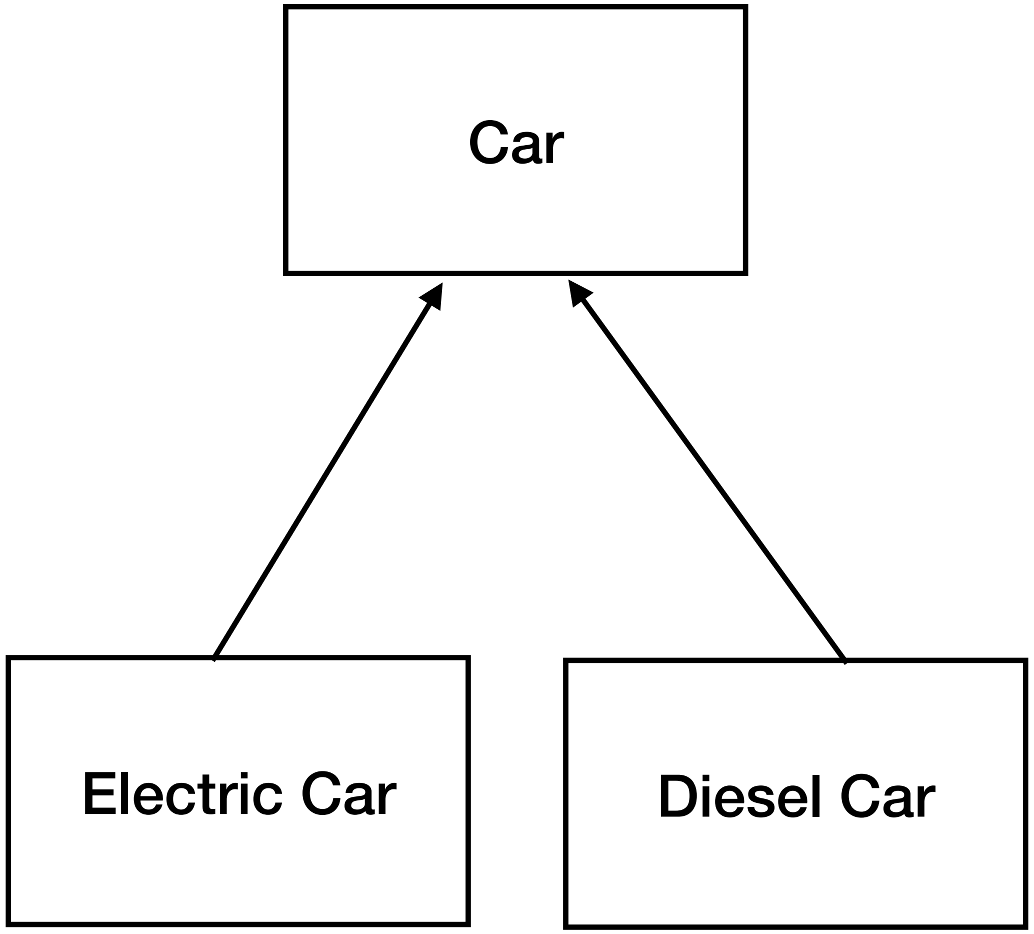
Outline

- Class and Objects
- Encapsulation and Access Control
- **Inheritance**
- Polymorphism
- Abstraction

Inheritance

- Lots of coding is repetitions.

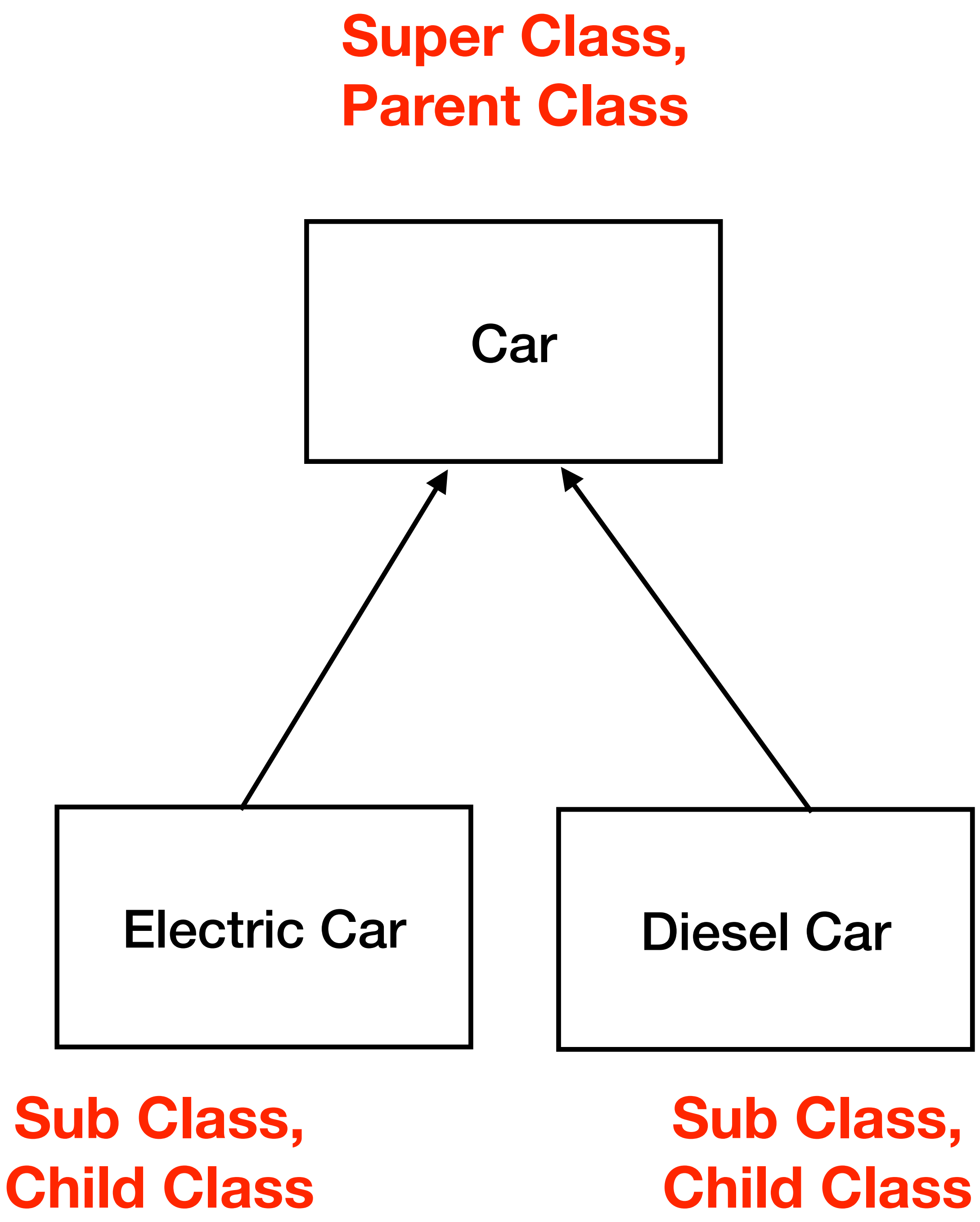
- **Inheritance** defines a relationship between classes with common attributes. Inheritance facilitates code reuse and result in code that is more readable and maintainable.



Inheritance

- Lots of coding is repetitions.

- **Inheritance** defines a relationship between classes with common attributes. Inheritance facilitates *code reuse* and result in code that is more *readable* and *maintainable*.



Super Class, Parent Class

Car

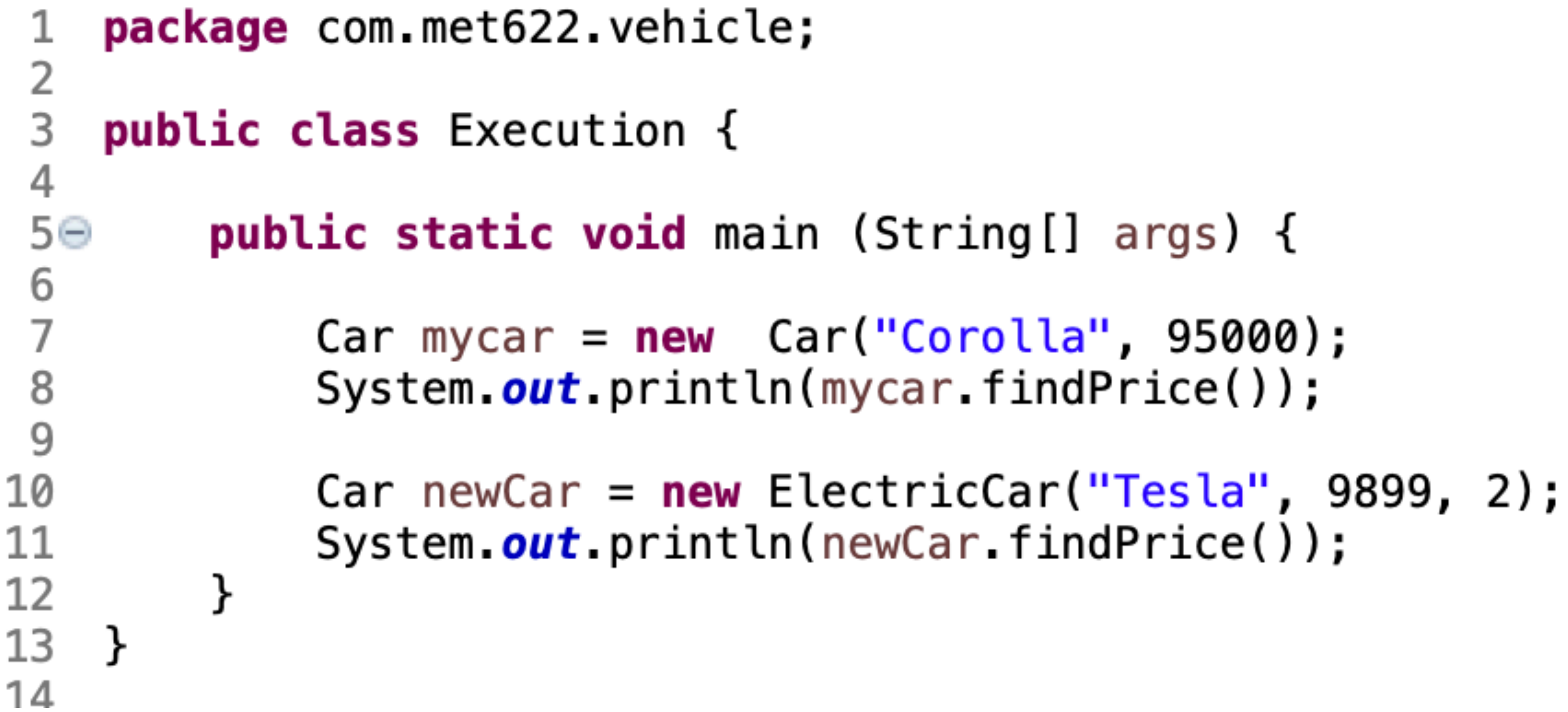
Electric Car

Diesel Car

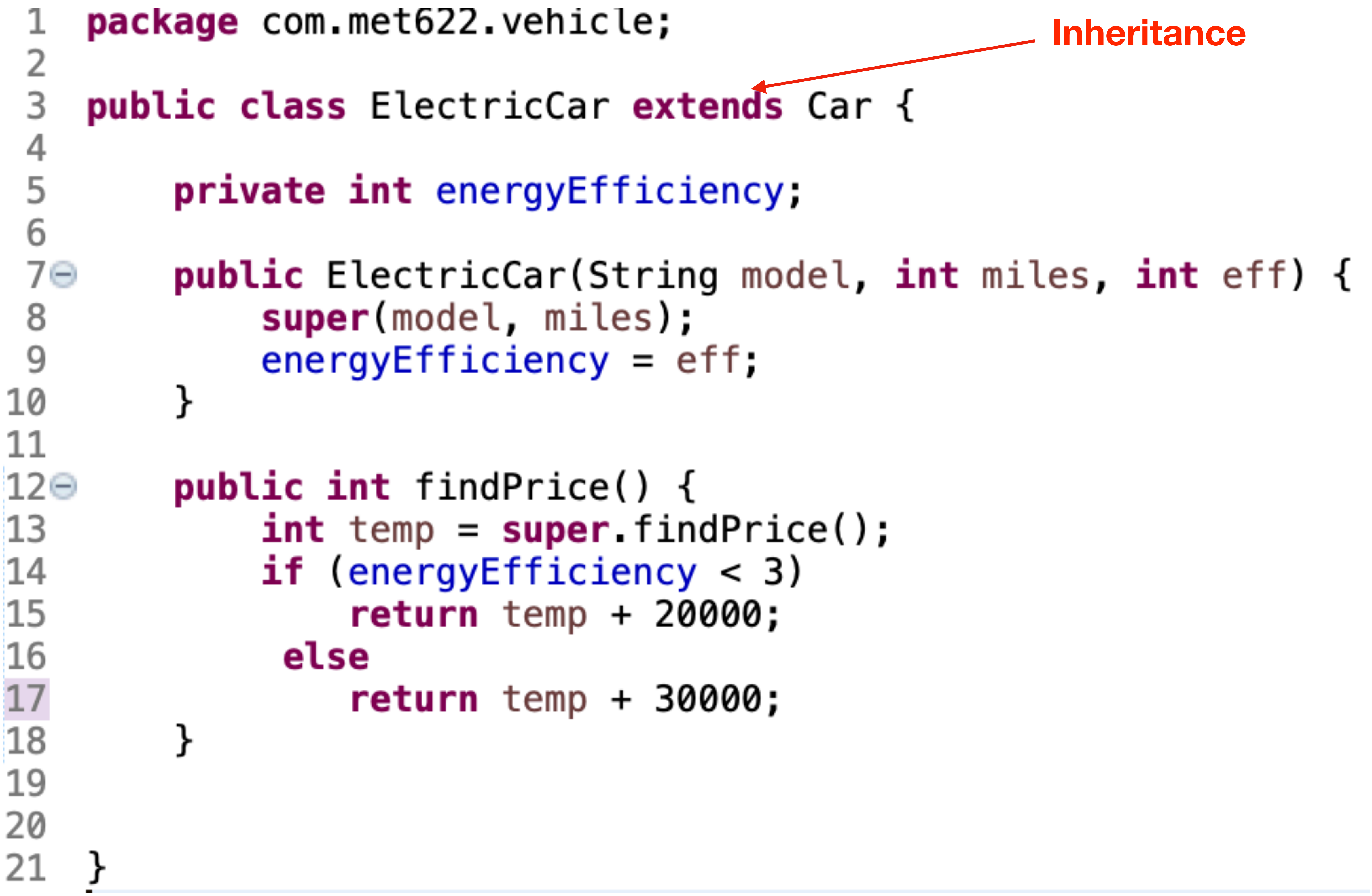
Sub Class, Child Class

Sub Class, Child Class

Inheritance Example

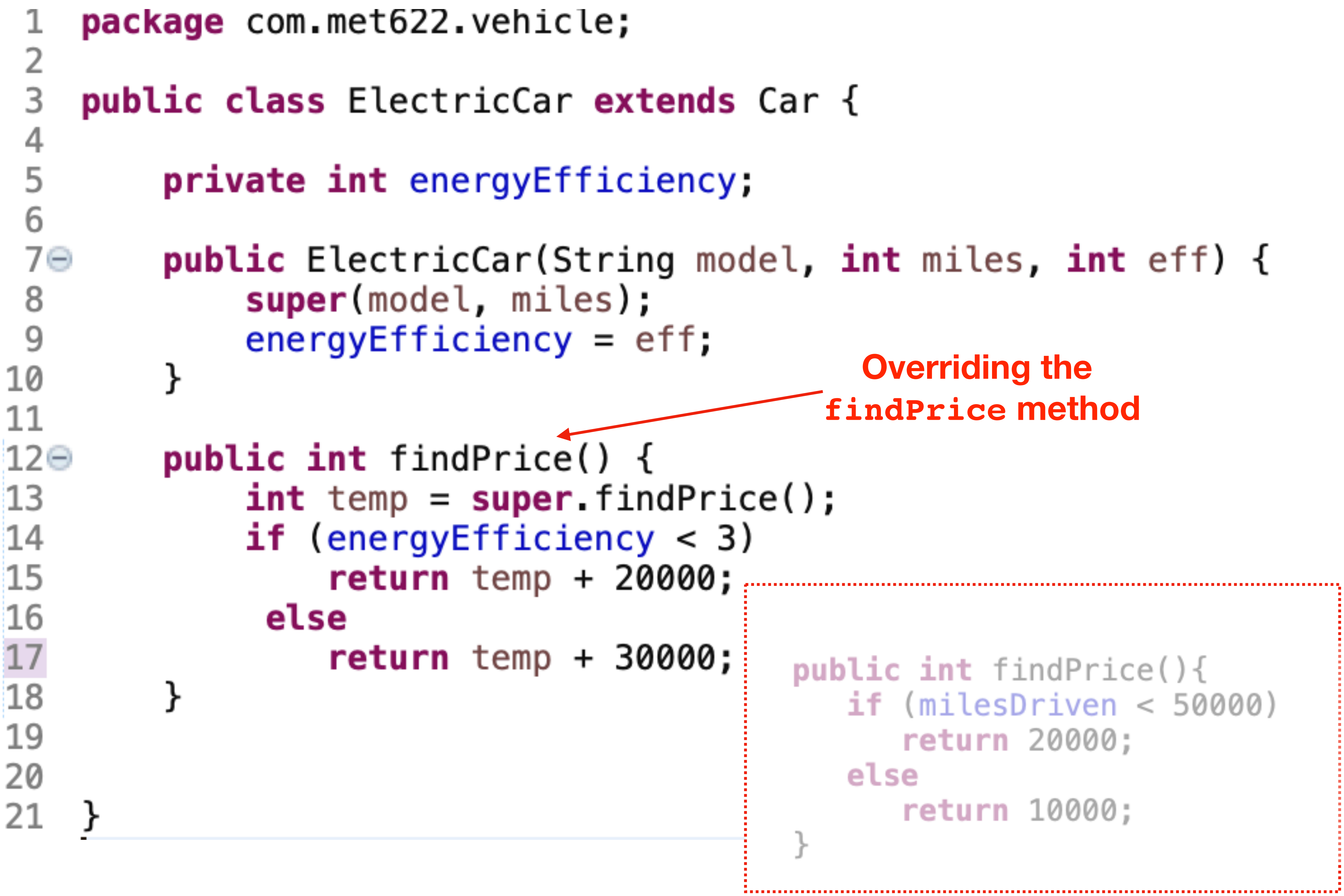


```java
package com.met622.vehicle;

public class Execution {

    public static void main (String[] args) {

        Car mycar = new  Car("Corolla", 95000);
        System.out.println(mycar.findPrice());

        Car newCar = new ElectricCar("Tesla", 9899, 2);
        System.out.println(newCar.findPrice());
    }
}

```



```java
package com.met622.vehicle;

public class ElectricCar extends Car {

    private int energyEfficiency;

    public ElectricCar(String model, int miles, int eff) {
        super(model, miles);
        energyEfficiency = eff;
    }

    public int findPrice() {
        int temp = super.findPrice();
        if (energyEfficiency < 3)
            return temp + 20000;
         else
            return temp + 30000;
    }


}
```

Inheritance



```java
package com.met622.vehicle;

public class ElectricCar extends Car {

    private int energyEfficiency;

    public ElectricCar(String model, int miles, int eff) {
        super(model, miles);
        energyEfficiency = eff;
    }

    public int findPrice() {
        int temp = super.findPrice();
        if (energyEfficiency < 3)
            return temp + 20000;
         else
            return temp + 30000;
    }


}
```

Overriding the `findPrice` method

```java
public int findPrice(){
    if (milesDriven < 50000)
        return 20000;
    else
        return 10000;
}
```

Inheritance

- A subclass **inherits everything** (all attributes and all methods with the exception of constructors) of its superclass has, and typically adds some (possibly zero) new attributes and/or some (possibly zero) new methods.

- A subclass can also provide a new definition to a method that it inherits; this is referred to as **overriding**.

- The ElectricCar class inherits method **`findPrice()`** from the Car class and then overrides it.

- Sometimes, while overriding a particular method, a subclass may retain the computation done in the method in the superclass, and then add something to it. This is achieved by a special method call "super".

Outline

- Class and Objects
- Encapsulation and Access Control
- Inheritance
- **Polymorphism**
- Abstraction

Polymorphism

- Polymorphism allows the compiler to be extended with new specialized objects being created while allowing **current part of the system** to **interact with a new object without concern for specific properties of the new objects**.

Writing Instrument
Specialized Writing Instruments

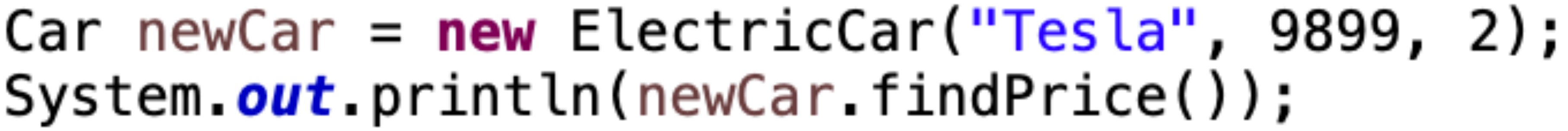
Source of this example: https://dev.to/charanrajgolla/beginners-guide---object-oriented-programming

Polymorphism

- Polymorphism is an ability that an object can take many form.

- A class that can pass “is-a” test is considered to be polymorphic.

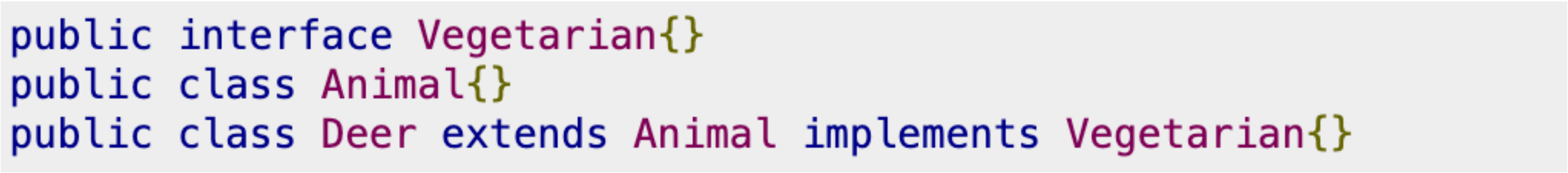
- e.g.



```
Car newCar = new ElectricCar("Tesla", 9899, 2);
System.out.println(newCar.findPrice());
```

- newCar is-a Car
- newCar is-a ElectricCar

Polymorphism



```
public interface Vegetarian{}
public class Animal{}
public class Deer extends Animal implements Vegetarian{}
```

- Deer is-a Animal
- Deer is-a Vegetarian
- Deer is-a Deer

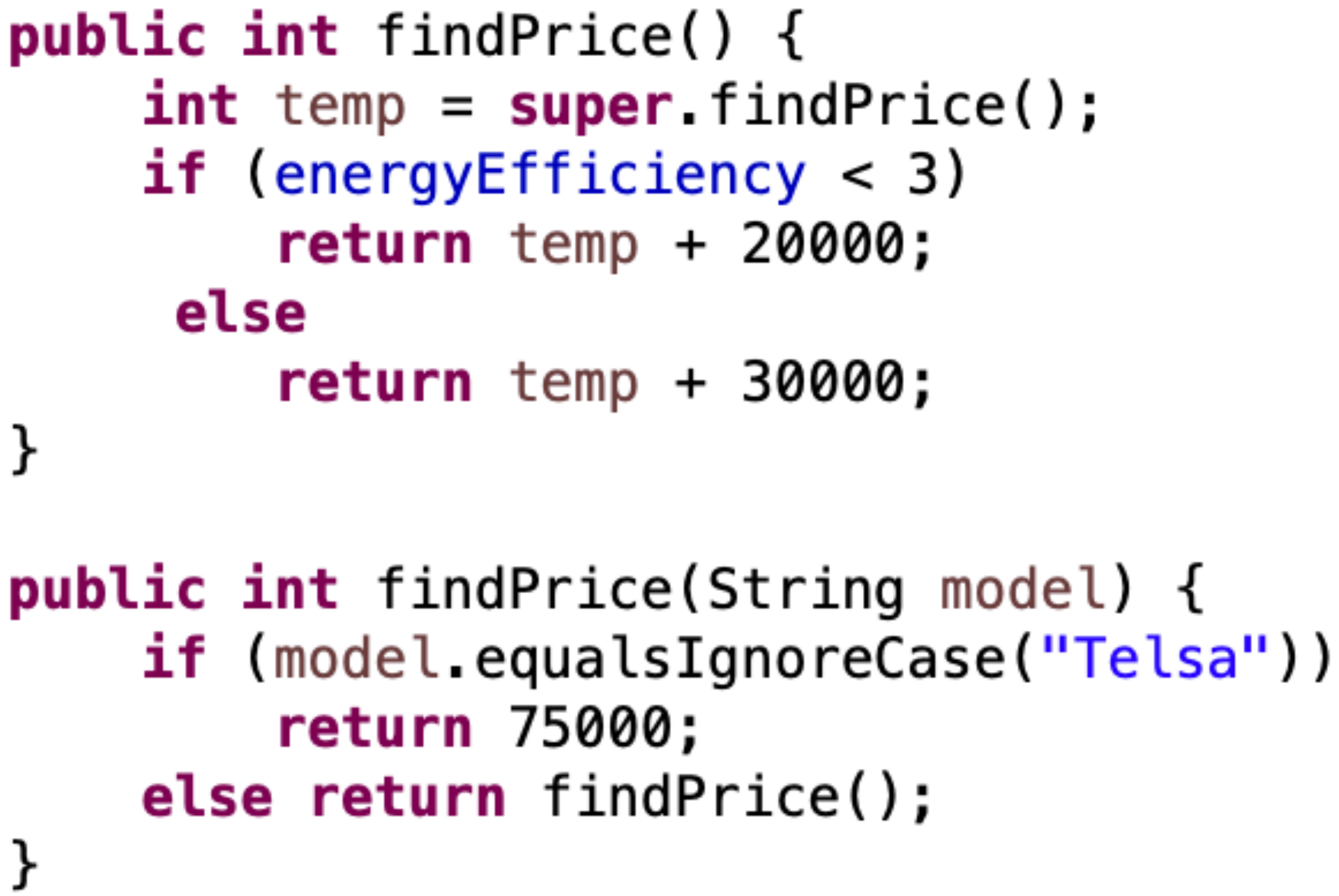
Inheritance causes polymorphism where behavior defined in the inherited class can be overridden by writing a custom implementation of the method. e.g. `findPrice` method in Car classes.

Source of this example: https://www.tutorialspoint.com/java/java_polymorphism.htm

Polymorphism

- Overriding
- Overloading

Overloading



```java
public int findPrice() {
    int temp = super.findPrice();
    if (energyEfficiency < 3)
        return temp + 20000;
     else
        return temp + 30000;
}

public int findPrice(String model) {
    if (model.equalsIgnoreCase("Telsa"))
        return 75000;
    else return findPrice();
}
```

Overriding - Overloading

- Overloading: When two or more methods in the same class has the same name but different parameters.

- Overriding: When we have two methods with exactly the same name and same parameters, but one method is inside the parent class and the other method is inside the child class. Their content is also different, otherwise we don't need to have a method described twice.

- When a method/variable is defined as **`final`** it can **not** be overridden. We define a final method/variable, when we need to guarantee a certain behavior in a class, disregard its child class.

- Variables are not possible to be overridden, and obviously overloading does not existed for variables.

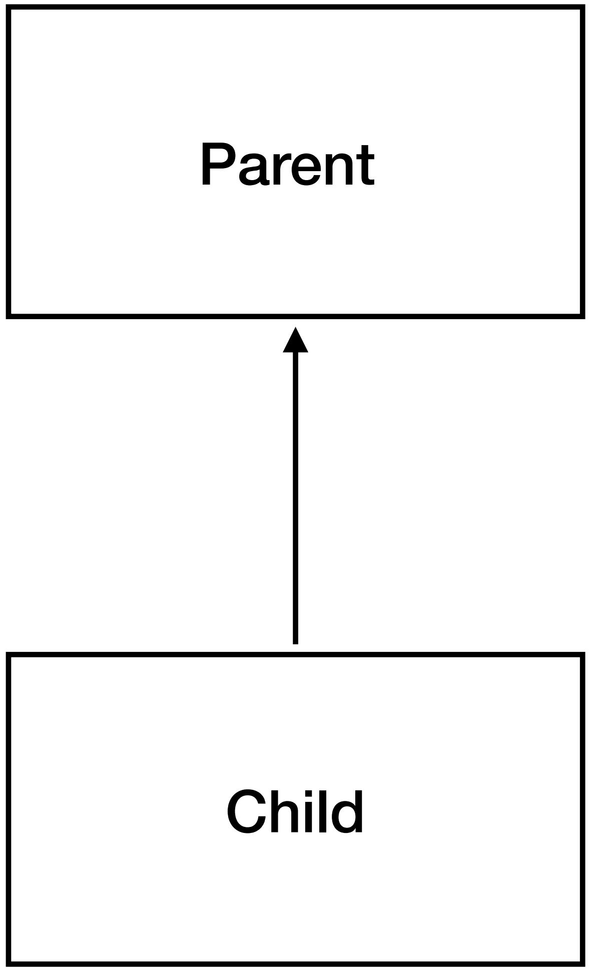
Downcasting and Upcasting

- Upcasting occurs when we cast a child class to the parent class.

- Downcasting occurs when we cast a parent class to the child class.

- Upcasting is always allowed but down casting causes a type check and it throws “`ClassCastException`”(if we do not specify the class type of parent as a child).

Downcasting and Upcasting



Upcasting:

```
Parent p = new Child();
```

Downcasting (raise error):

```
Child c = new Parent();
```

Correct:

```
Parent p = new Child();
Child c = new Child();
```

Outline

- Class and Objects
- Encapsulation and Access Control
- Inheritance
- Polymorphism
- **Abstraction**

Abstraction

- Abstraction is the process of hiding implementation details from the user.

  e.g. `cluster(dataset, algorithm name)`

  We do not need to know how clustering works. We only need to give the dataset and the algorithm name.

Abstract Class

- `'abstract'` is a keyword in Java and we can define a class as abstract.

- An abstract class can not be instantiated. But, its inherited class can.

- If a class has one abstract method, its class should be also abstract.

```
public abstract class TestAbs {
  …
TestAbs a = new TestAbs();
```

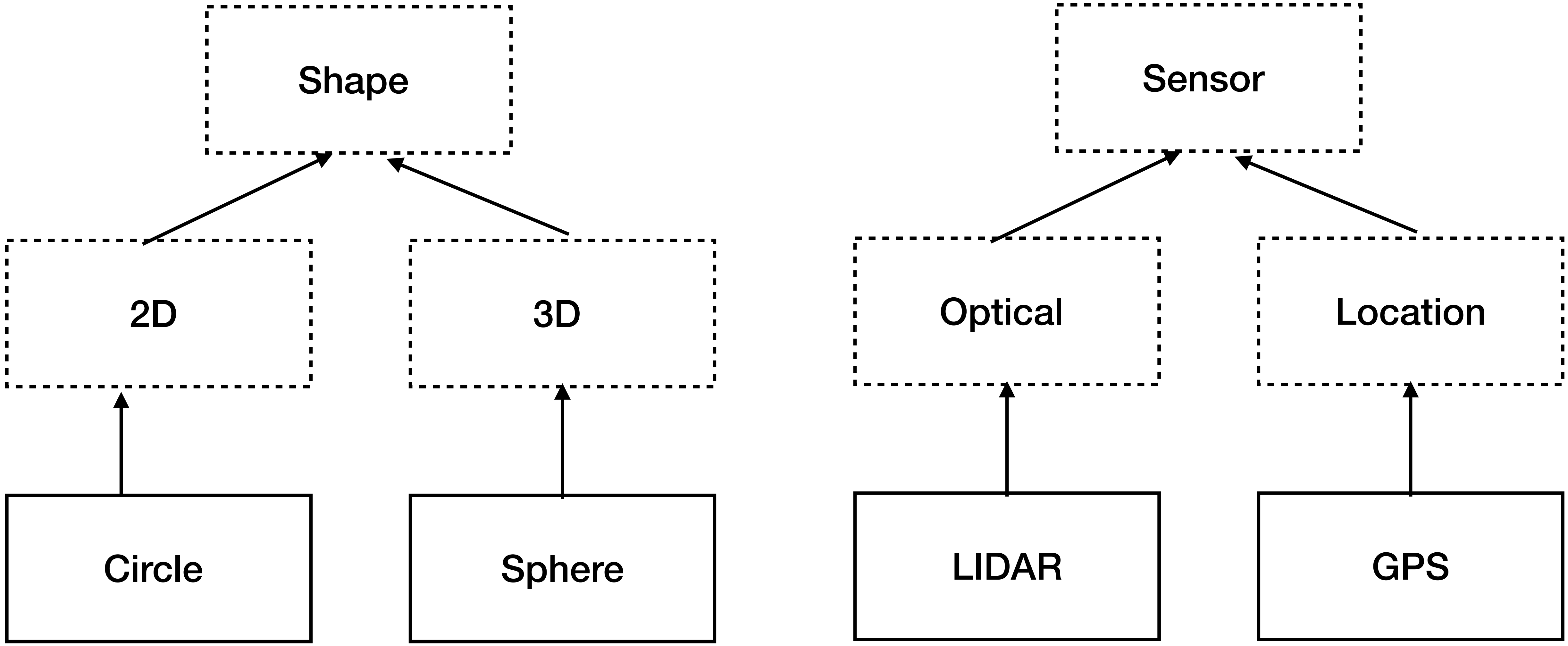
Error

- An inherited class from abstract class can be instantiated.

```
public class ChildAbs extends TestAbs{
  …
ChildAbs a = new ChildAbs();
```

Correct

Abstract Class Examples



An abstract class is a concept that is required to be implemented by its subclasses

Abstract Class

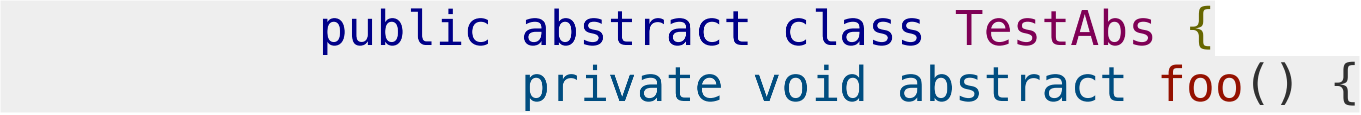
- An abstract method should not contains the body, it only has method names.

- An abstract class can be extended only and not instantiated.

- An abstract class/method can **not** be **private** or **final**.

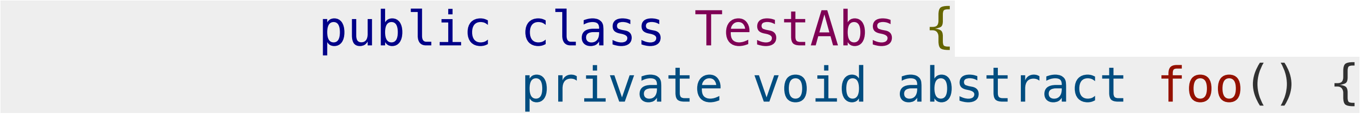
- A non-abstract child class that extends an abstract method must implement (override) all of its method without exception.

- An abstract class **can** contain non-abstract methods. But a non-abstract class **cannot** contain abstract method.



```
public abstract class TestAbs {
       private void abstract foo() {
```

Correct



```
public class TestAbs {
       private void abstract foo() {
```

Error

Interface

- Java supports single inheritance, i.e. a class can extend only one class. To handle this we use `interface`.

- Interface is similar to an abstract class (it can not be instantiated), but it can contain only constants, abstract methods and static methods. It can not contain non-constant fields, but abstract class can contain non-constraint fields.

- In other words, interface is similar to class, but it includes abstract methods and all of its method should be public.

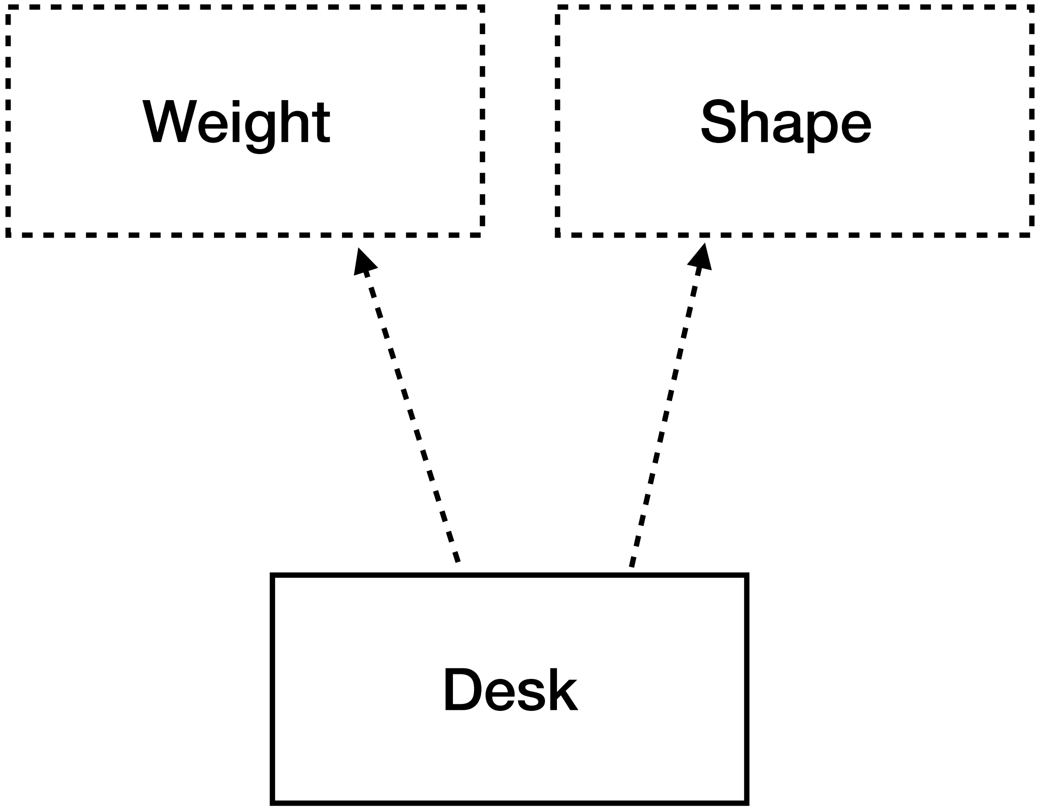
- We cannot instantiate an Interface.

- Interface does not have a constructor.

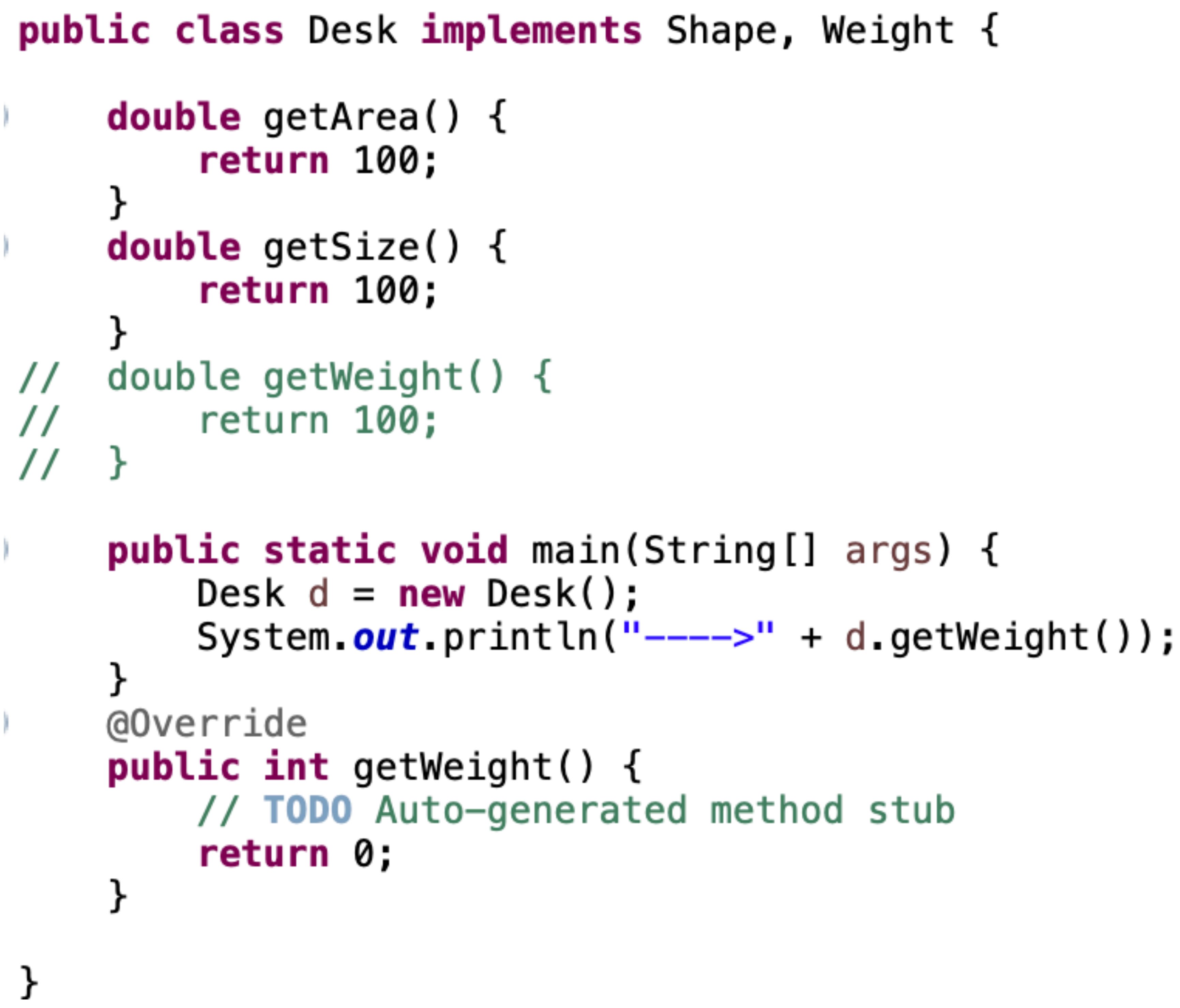
- All fields in the “interface” should be defined as “static” and “final”.

- All methods in the “interface” must be “public” and “abstract”.

Example

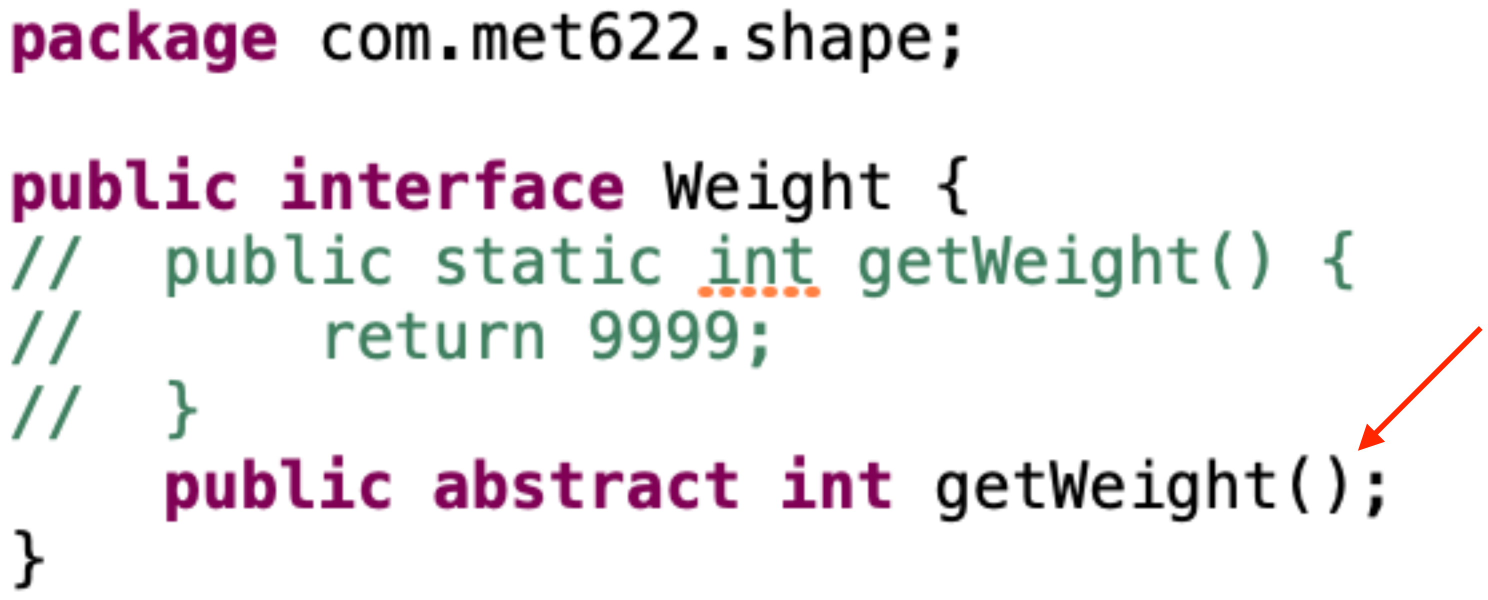


Example



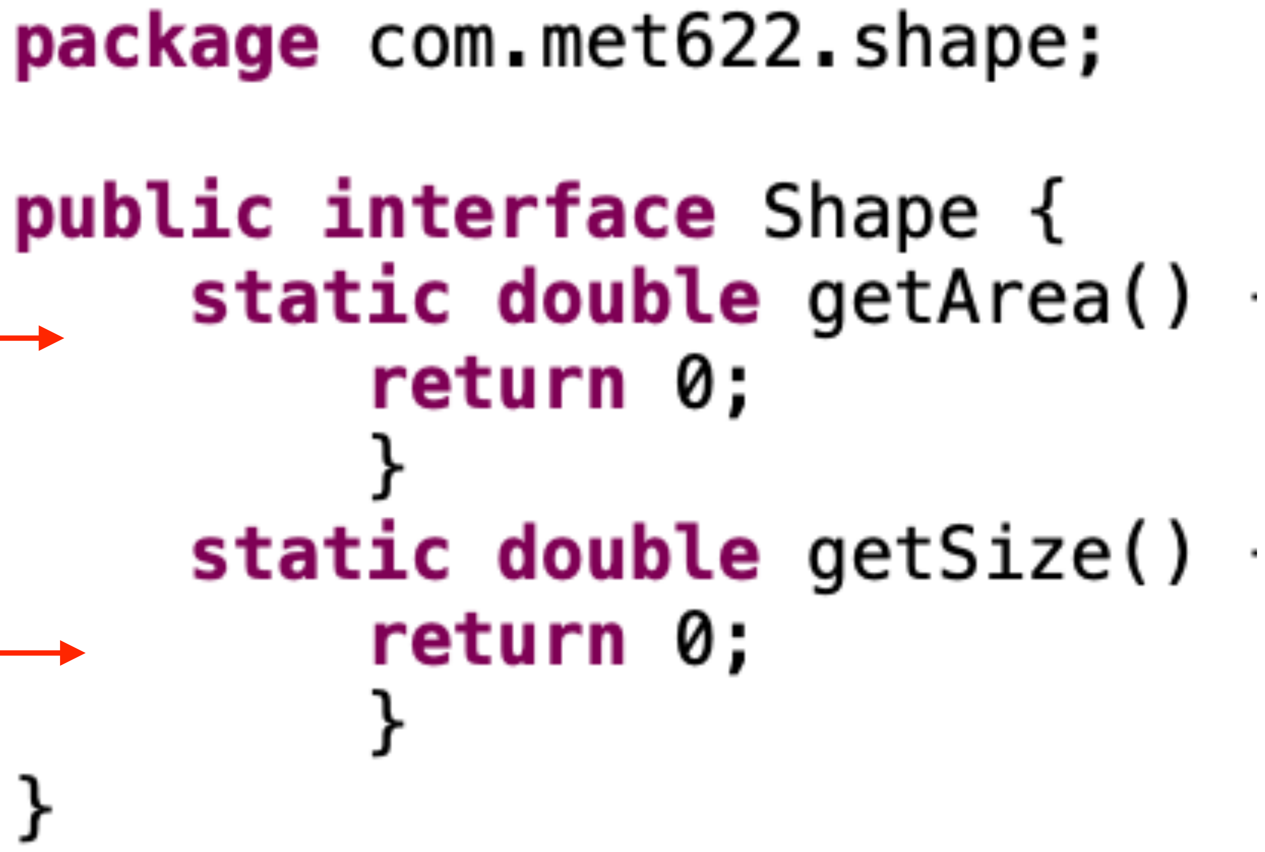
```
public class Desk implements Shape, Weight {

    double getArea() {
        return 100;
    }
    double getSize() {
        return 100;
    }
//  double getWeight() {
//      return 100;
//  }

    public static void main(String[] args) {
        Desk d = new Desk();
        System.out.println("---->" + d.getWeight());
    }
    @Override
    public int getWeight() {
        // TODO Auto-generated method stub
        return 0;
    }

}
```

Example



```
package com.met622.shape;

public interface Weight {
//  public static int getWeight() {
//      return 9999;
//  }
    public abstract int getWeight();
}
```

Correct



```
package com.met622.shape;

public interface Shape {
    static double getArea()
        return 0;
    }
    static double getSize()
        return 0;
    }
}
```

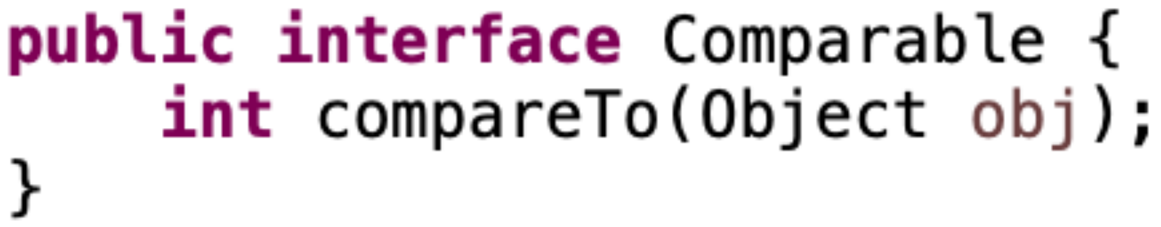
Incorrect
but not error

Incorrect
but not error

Comparable Interface

- Sometimes we define an object with lots of properties and we need our own comparison policy.

- For example, assume we have the class ‘Athlete’ and we need to compare two athletes together. The comparison should be done based on both “strength” and “speed”.



```
public interface Comparable {
    int compareTo(Object obj);
}
```

Abstraction vs Encapsulation

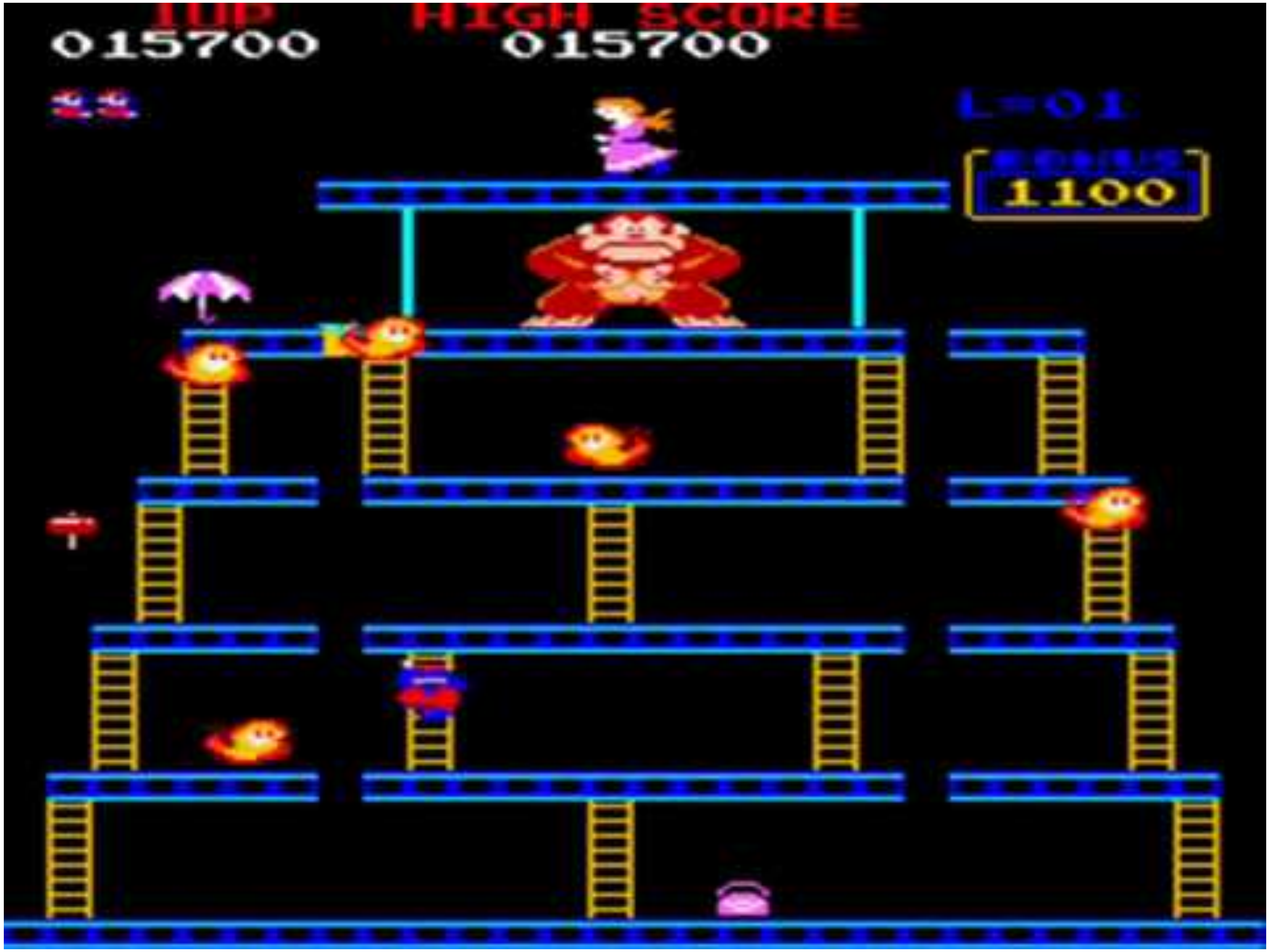
Abstraction	Encapsulation
It focuses on removing unnecessary information.	It focuses on keeping the data safe from outside access and misuse
We define it by using **Abstract** and **Interfaces** keywords	We define it by using Access Control Keywords, **public**, **private** and **protected**.
Abstraction focuses on the **design.**	Encapsulation focused on the **implementation.**

Global vs Local Variable

```
public class CLS {
    String x = new String (“variable 1”);
    String y = new String (“variable 2”);
    public void methodB (String x){
        System.out.print(x);
    }
    public void methodC( ){
      System.out.print(x);
    }
    public void methodA( ) {
        String x = new String(“somevariable”);
        System.out.print(x);
    }
}
```

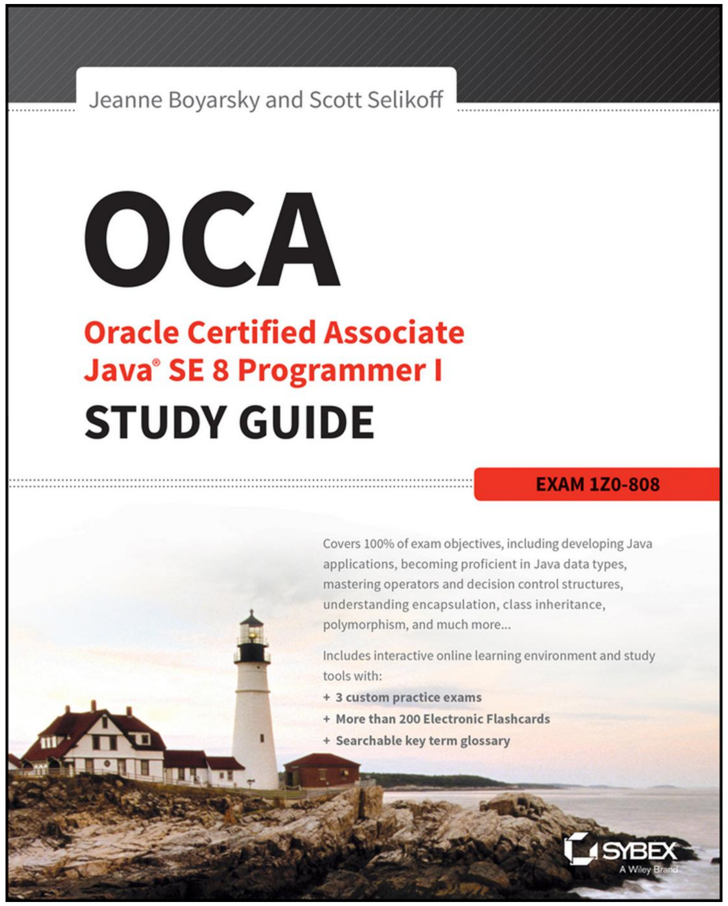
Lets do an Example for Object Oriented Game Design

Homework 1



References

- https://www.tutorialspoint.com/java



Jeanne Boyarsky and Scott Selikoff

OCA

Oracle Certified Associate
Java® SE 8 Programmer I

STUDY GUIDE

EXAM 1Z0-808

Covers 100% of exam objectives, including developing Java applications, becoming proficient in Java data types, mastering operators and decision control structures, understanding encapsulation, class inheritance, polymorphism, and much more...

Includes interactive online learning environment and study tools with:

+ **3 custom practice exams**
+ **More than 200 Electronic Flashcards**
+ **Searchable key term glossary**

SYBEX
A Wiley Brand