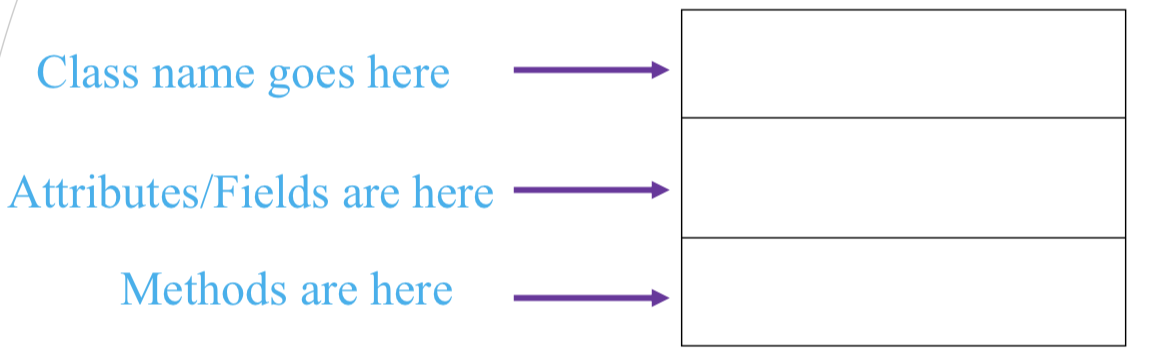
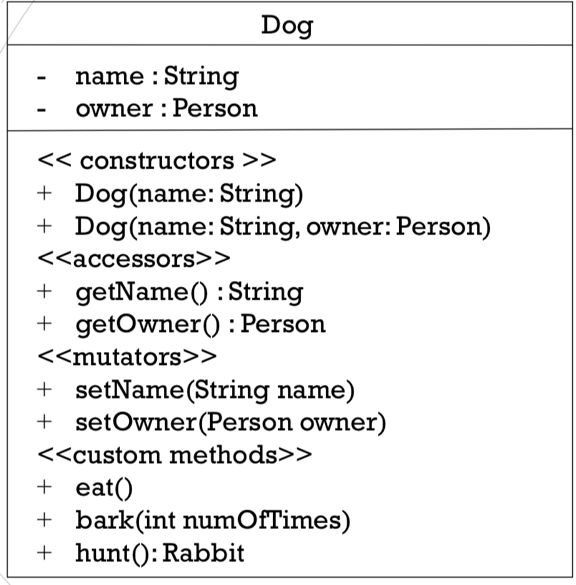
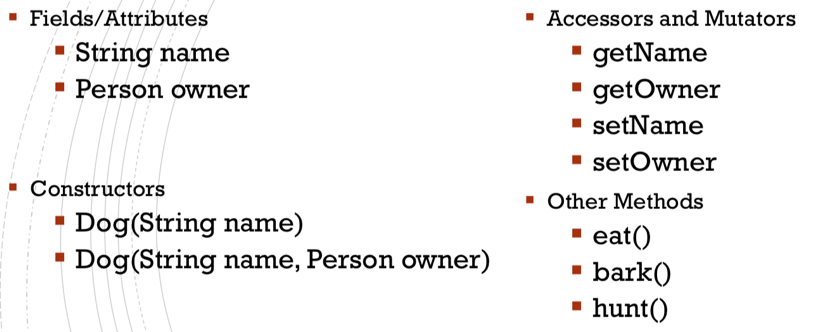
# COMP250-OOD basic

UML Diagram

*Unified Modeling Language (UML) provides a set of standard diagrams for graphically depicting object-oriented systems.*



Example: Dog Class



Packages

*A package is a group of classes. (Each class is referred to as a package member)*

To define a package we write at the top of our class file the following statement.



There are two main rules related to files’ and folders’ names in Java:

1.The name of the class must match the name of the file (with .java added)

2.The folder path must match exactly the package name

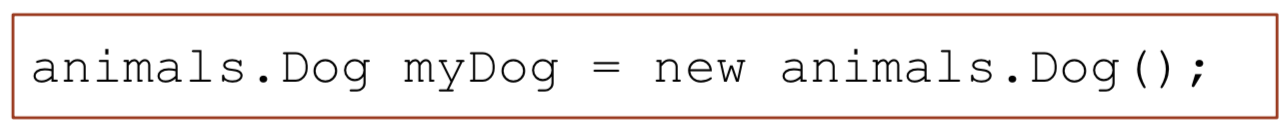
–except that each period is actually a “slash” (i.e. a subfolder)

The Java compiler automatically imports two entire packages for each source file:

* The java.lang package
* The current package

If you want to use a package member from **outside** its package, you must instruct your program where to find that class. You can do this in 3 ways:

1. Specify the entire path whenever you use such class.



2. Import the package member.

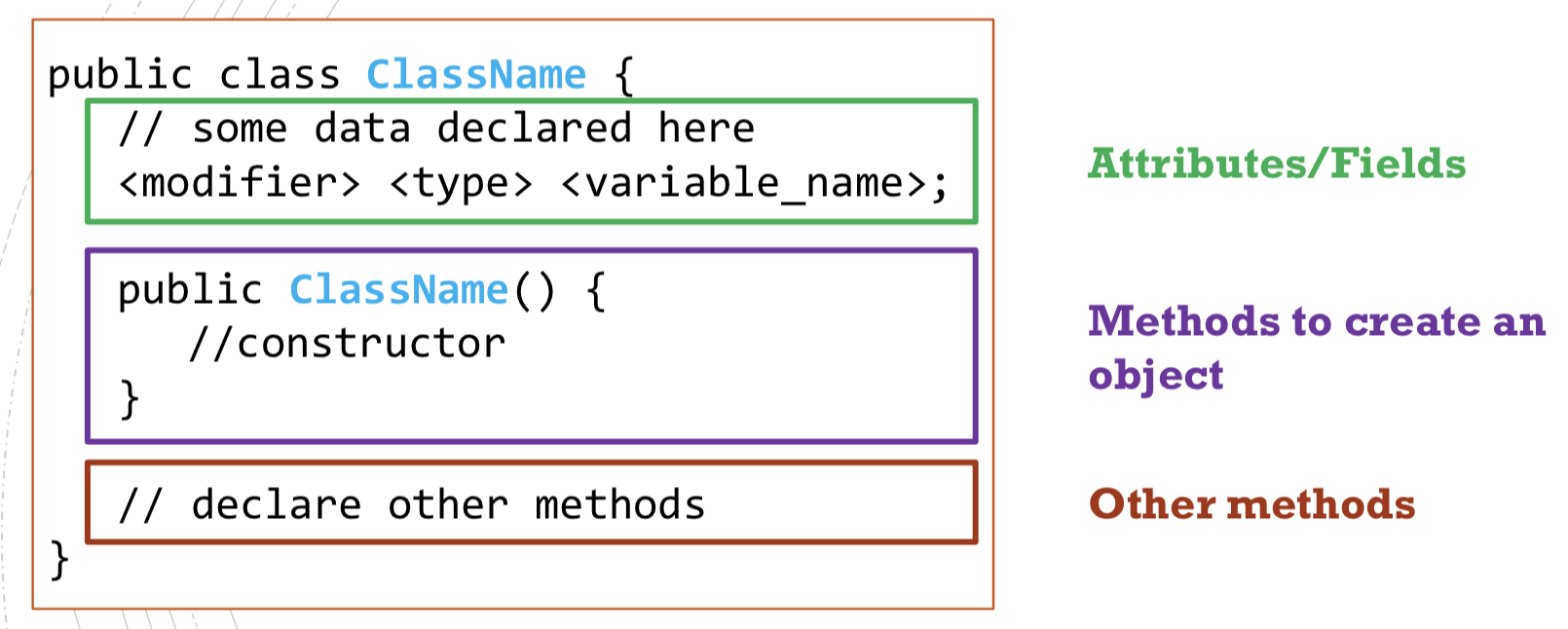


3. Import the entire package

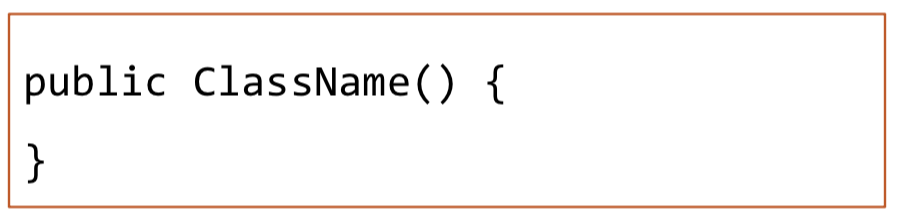


Class and Objects

*Each time we define a class, we create a new object type with the same name.*

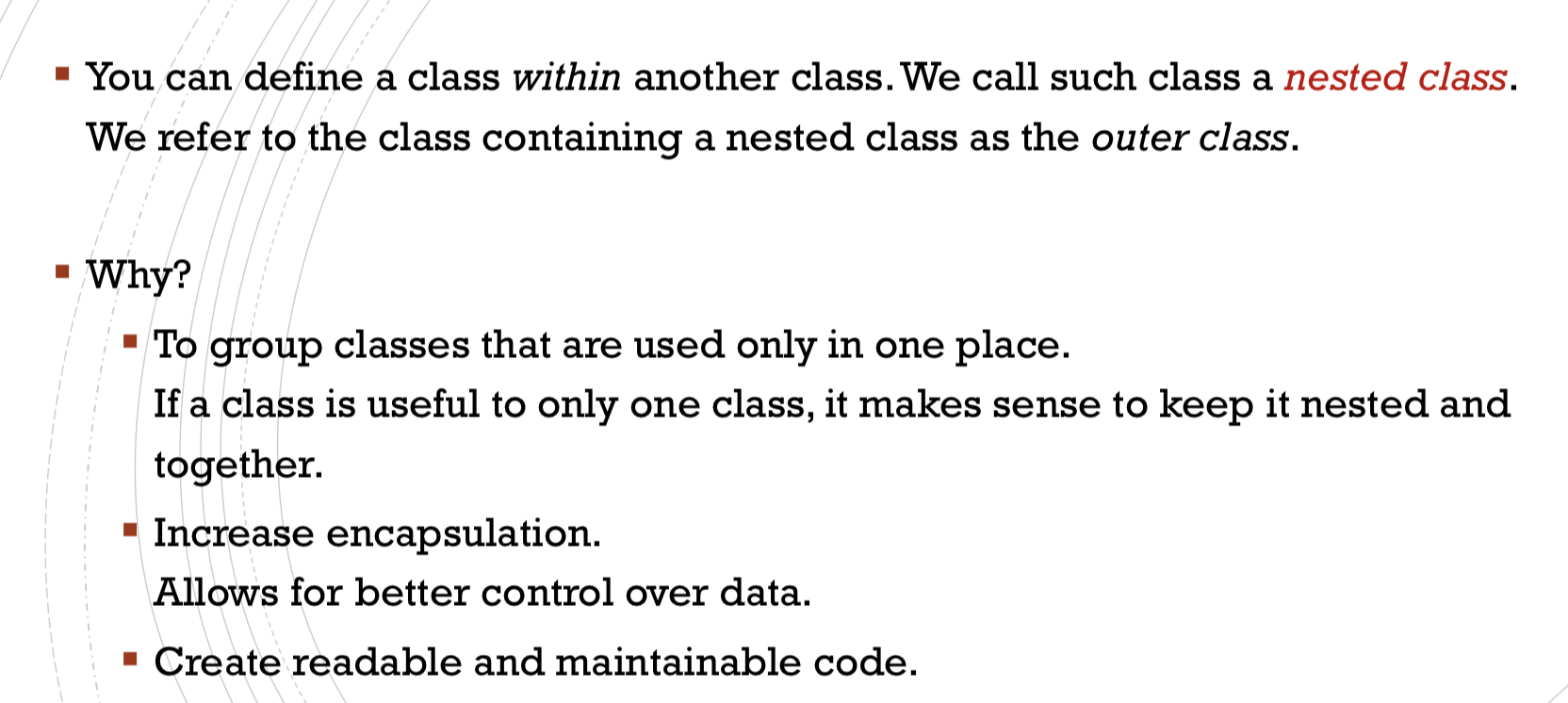


* Default constructor



But if you write your own constructor, you don’t have access to the default constructor.

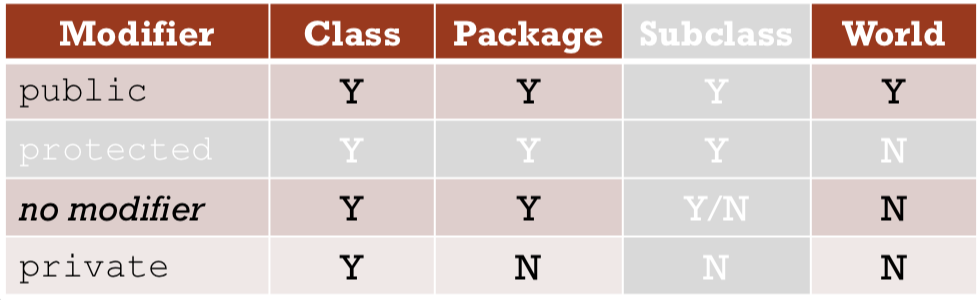
* Nested Classes



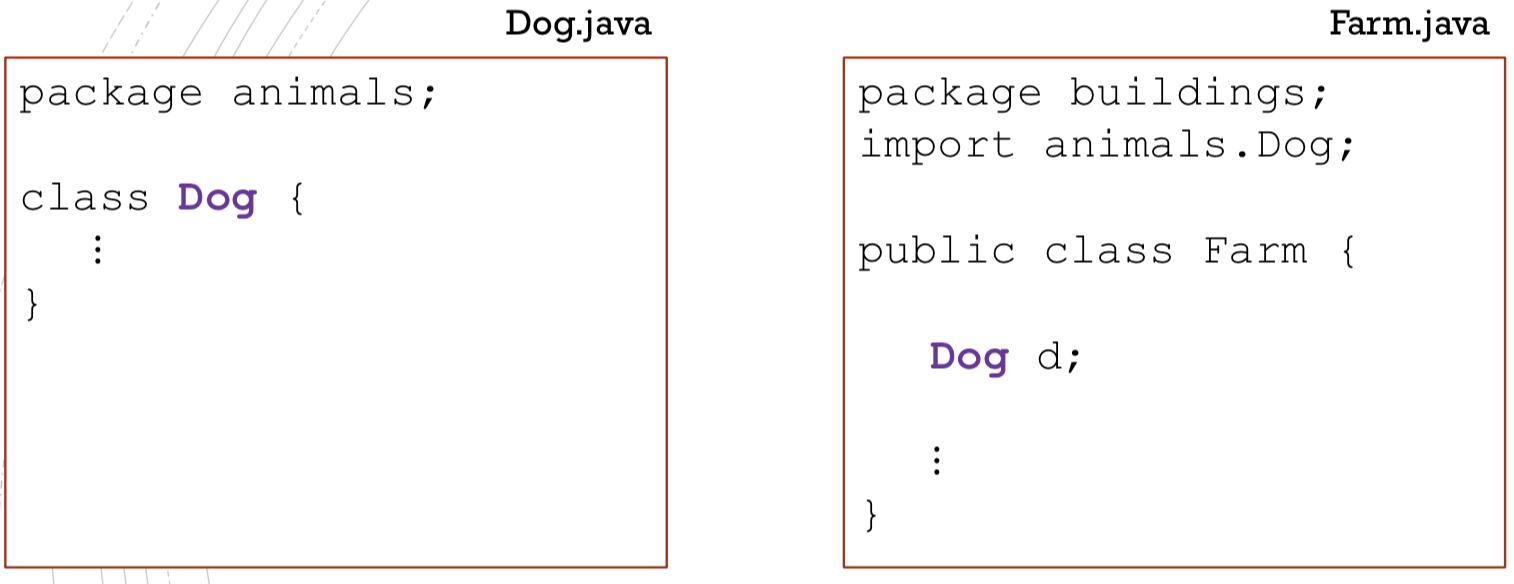
Modifiers

Modifiers are keyword added to class/method/variable’s definition to change their meaning.

Access Control Modifiers

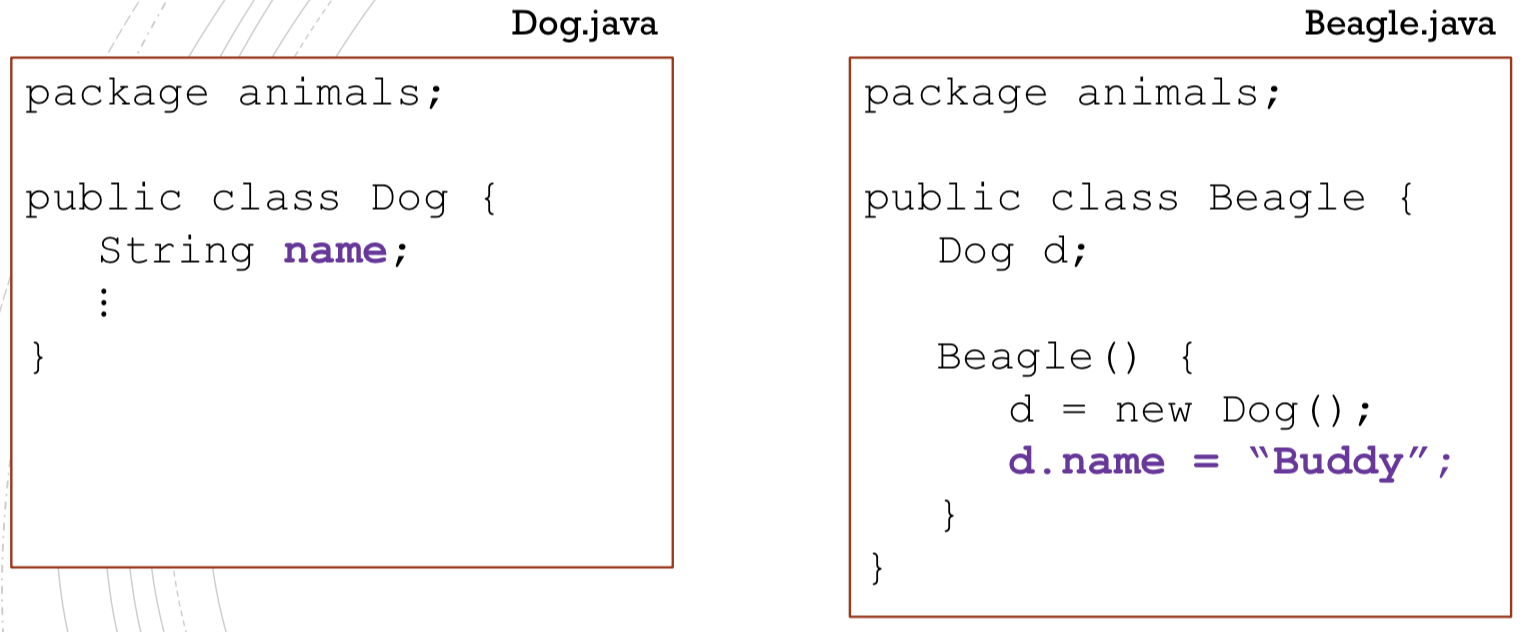


Note: Outer classes can only be declared public or package private.



NO

The class Dog is visible only within its package



YES

The field name is visible within its package

Encapsulation

*Process of wrapping data and the code acting on that data in one unit. The idea is to better control the data.*

* Make all the fields private
* Provide getters and setters as needed.

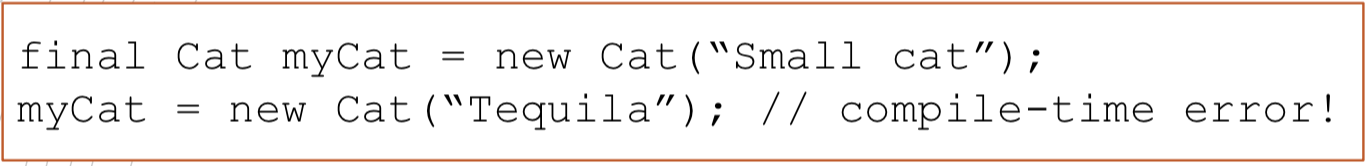
Non-access Modifiers

* static (Fields, methods, and nested classes)

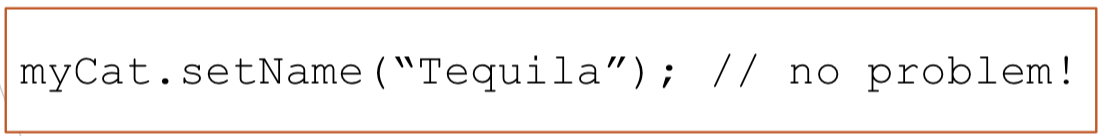
*When a class member is declared to be static, then it “belongs” to the entire class and not to a specific instance (object).*

* final (Variables, methods, and classes)

*If a variable is final, its value can never be changed after it has been assigned.*



However, you can change the object that myCat points at, without changing myCat’s value.



Final fields must be initialized! (Otherwise compile-time error)

* If the class has a final instance variable (i.e. a final non-static field), you must initialize it in every constructor!
* If the class has a final class variable (i.e. a final static field), you should initialize it in place(on the same line of the declaration) or in a Static Initializer Block (we might talk about this in the future).
* abstract (Methods and classes)

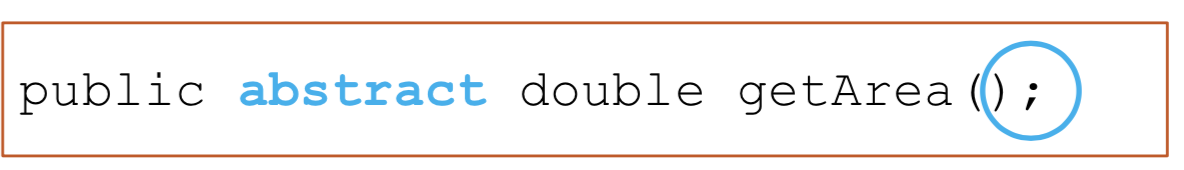
*If you want a class to contain a particular method, but you would like the implementation of this method to be specified by the subclasses, then you can declare the method to be abstract.*

* abstract method

An abstract method is a method that is declared without implementation:

The method has no body!

Instead of the curly braces, we use the semicolon at the end of the header.



* The class containing it must be also declared abstract.
* Every subclass of the current class MUST either override the abstract method or declare it itself as abstract.
* abstract class

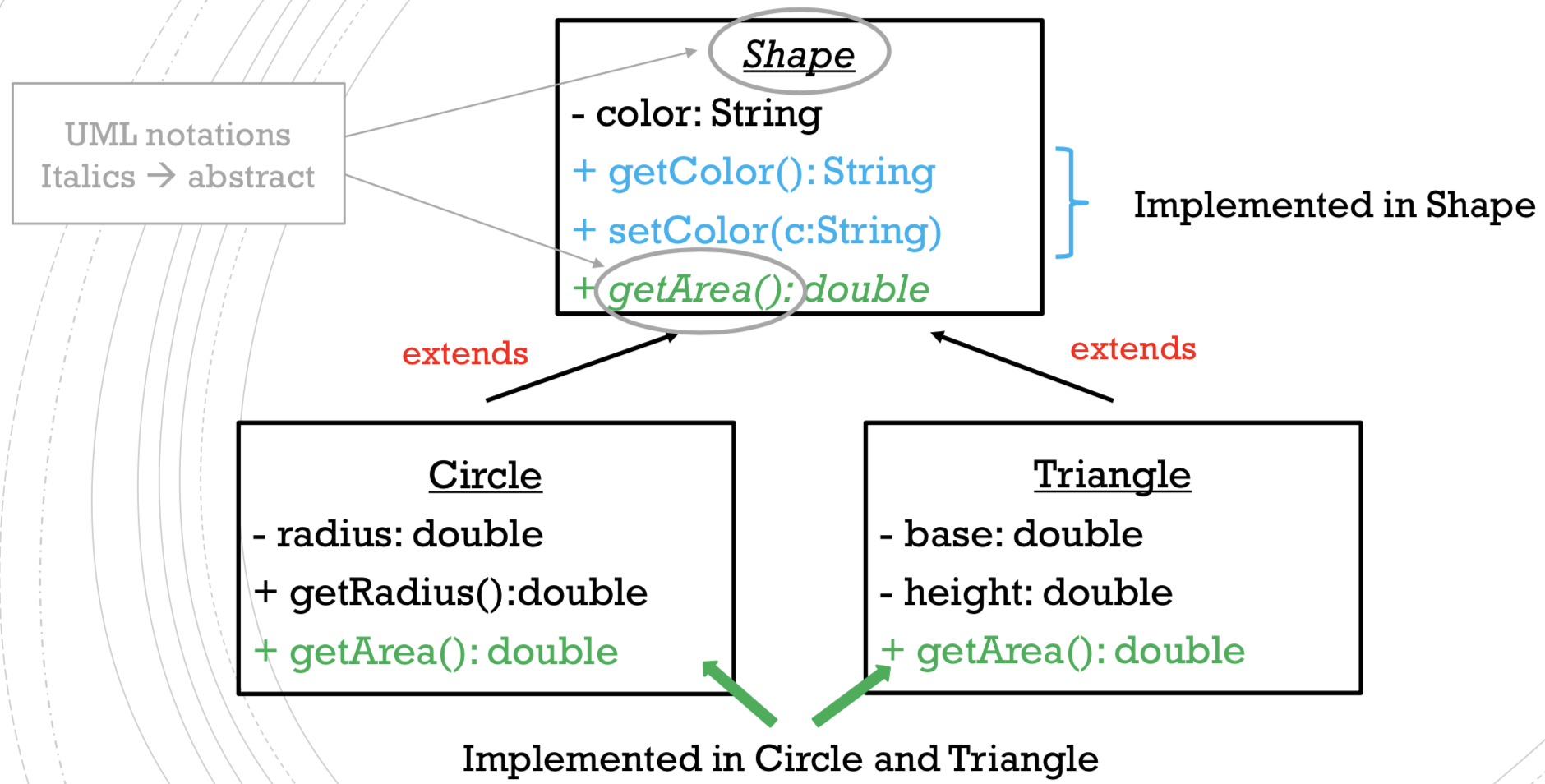
An abstract class must be declared with an abstract keyword.

It can have abstract and non-abstract methods.

It cannot be instantiated, but we can define constructors. These constructors are called when an instance of a subclass is created.

It can have constructors and static methods.

It can have final methods which will force the subclass not to change the body of the method



Inheritance

*In java, classes can be* ***derived*** *from other classes.*

*A class that is derived from another class is called a* ***subclass****. The class from which the subclass is derived is called a* ***superclass****.*

*A subclass inherits all public(or protected) fields and methods from its superclass. Constructors are the only thing that a subclass does not inherit.*

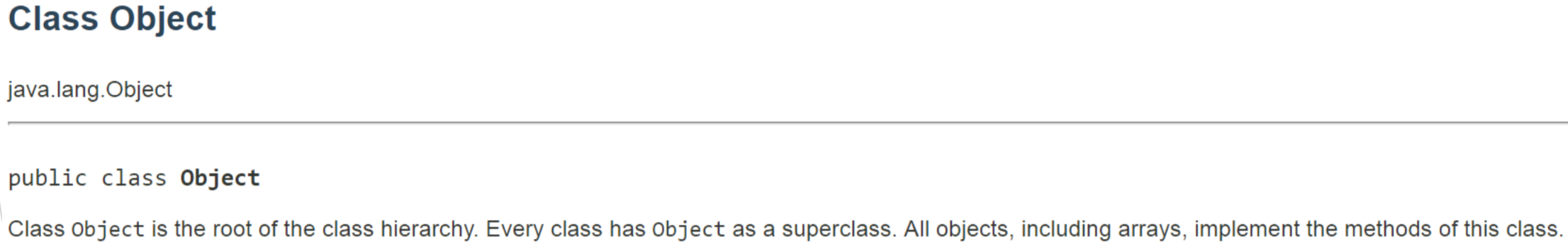
A class can extends class if and only if the latter is **visible from where the former is located**. (All public classes can be extended even across packages)

* A subclass cannot reduce the visibility of an inherited method. The visibility can only be increased.
* A nested class is not a subclass. Outer and inner classes have access to all fields and methods of each other. (Details are out of the scope of this course)
* A class that has been declared **final** cannot be extended---compile-time error

A method that has been declared **final** cannot be overridden---compile-time error

Object Class

Object is the only class in java without a superclass. All other classes have one and only one direct superclass. In the absence of any other specific superclass, every class is implicitly a subclass of object.



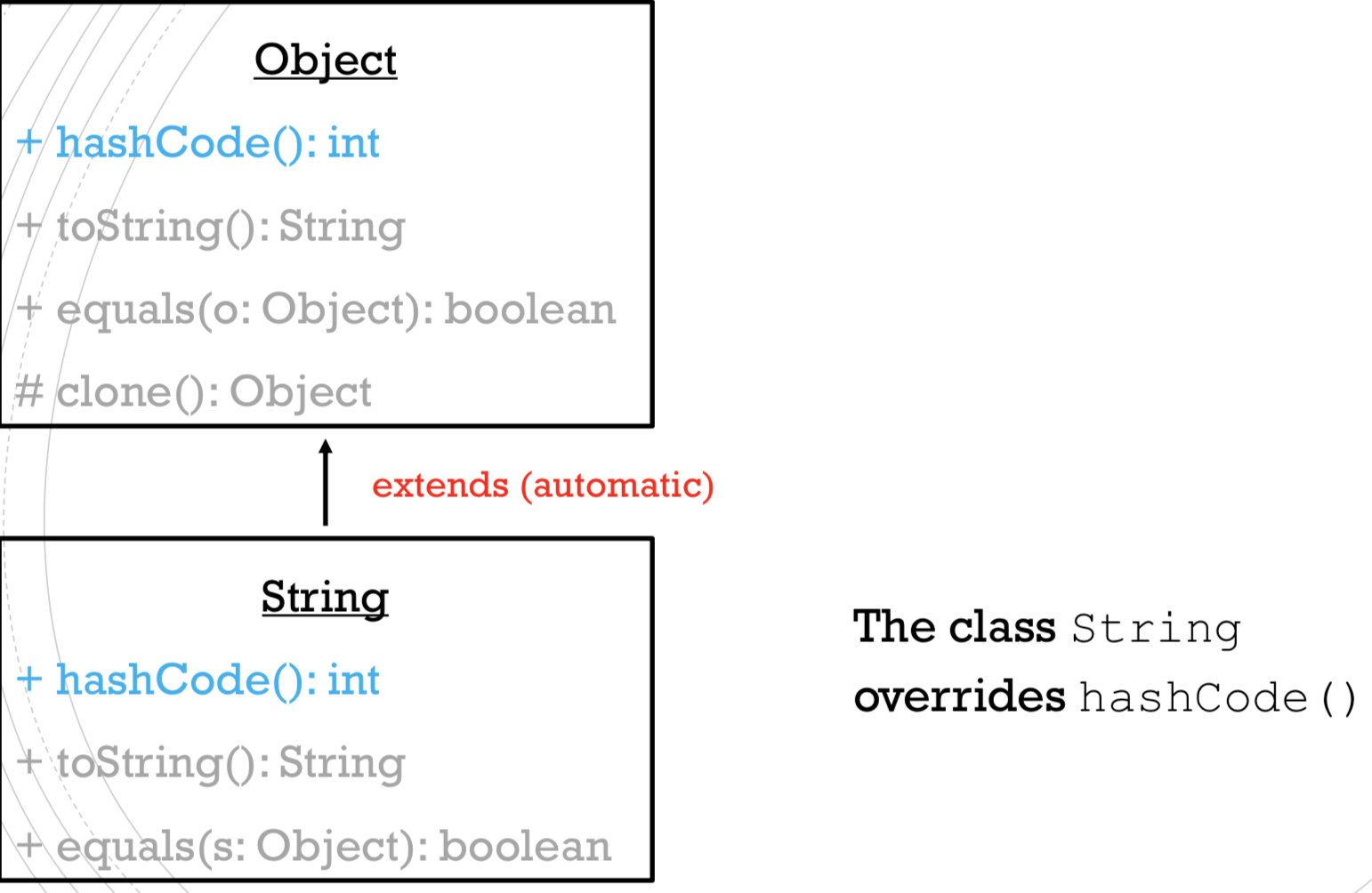
* hashCode()

*It returns a 32 bit integer associated to this object.*

“typically implemented by converting the internal address of the object into an integer, but this implementation technique is not required by the Java™ programming language”.

If o1.equals(o2)is true, then o1.hashCode()==o2.hashCode() should also be true.

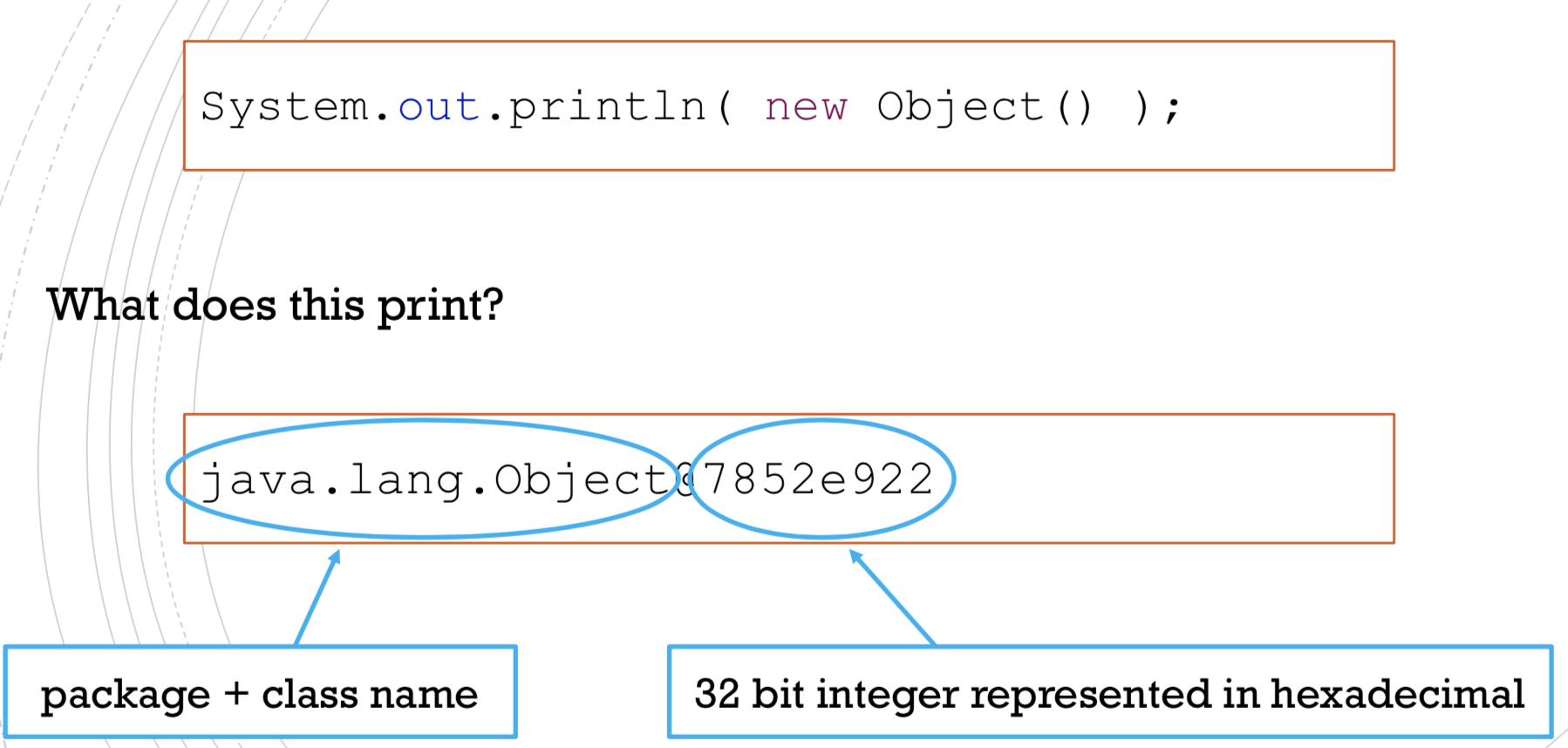
Note that the converse does not need to hold!



* toString()

*Returns a string representation of the object.*

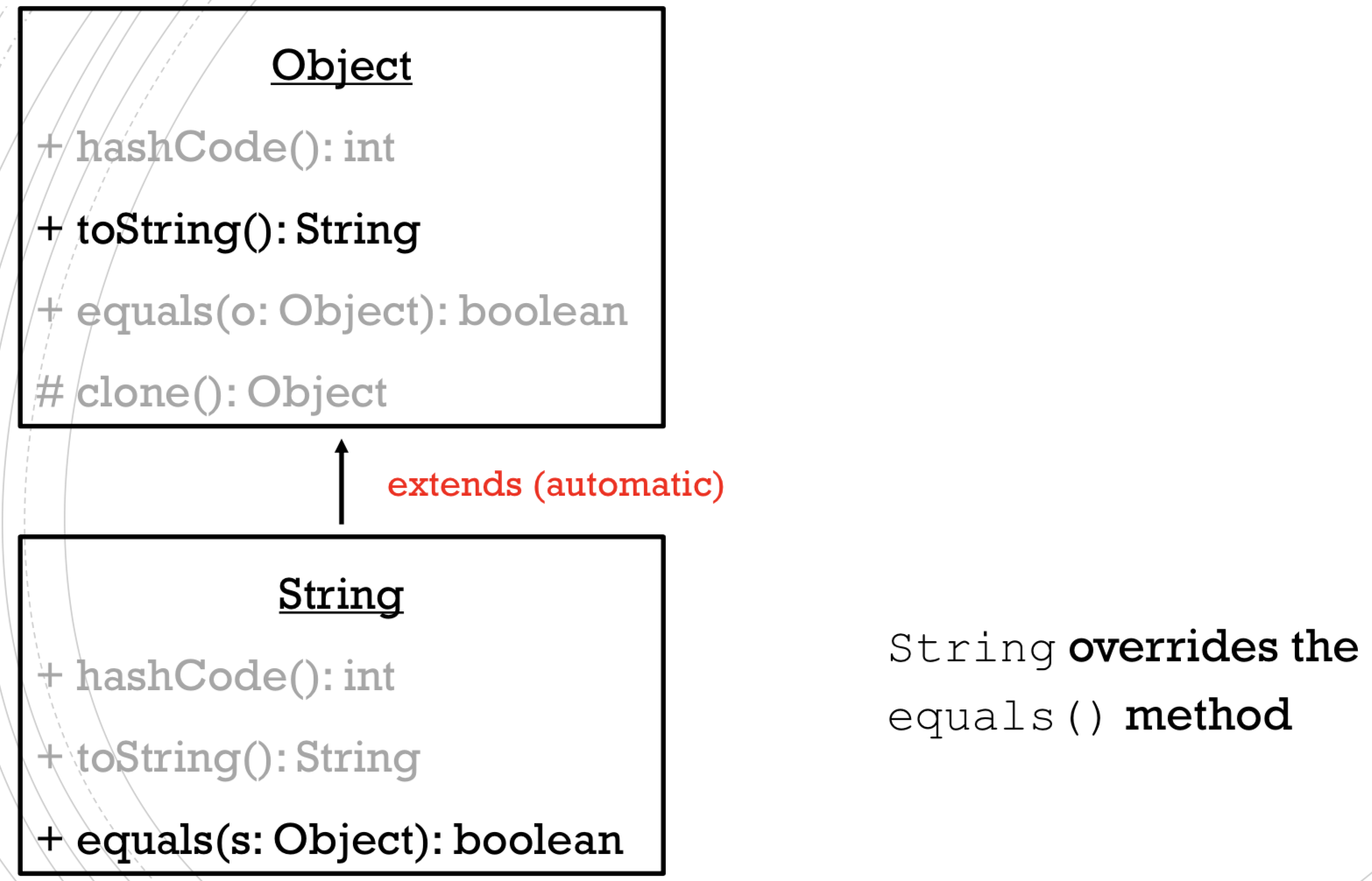
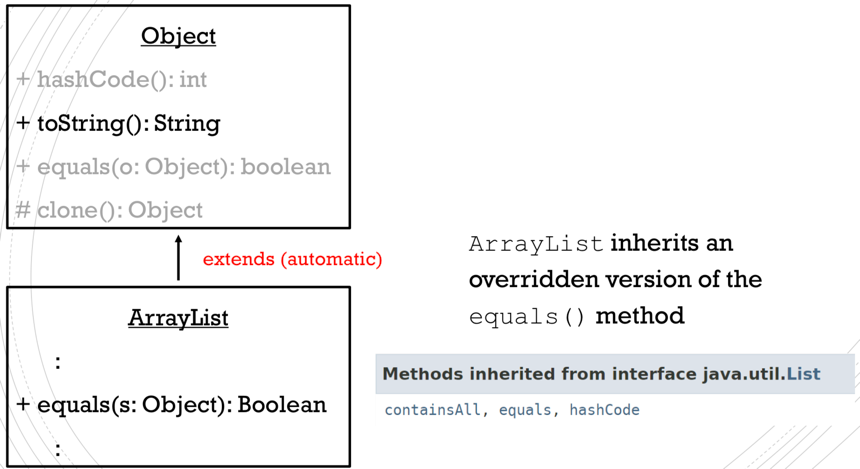
For class Object, it returns **className + “@” +Integer.toHexString(hashCode())**



It is recommended that all subclasses override this method.

* equals()

For any non-null reference values obj1 and obj2, obj1.equals(obj2) returns true if and only if obj1 == obj2 has value true



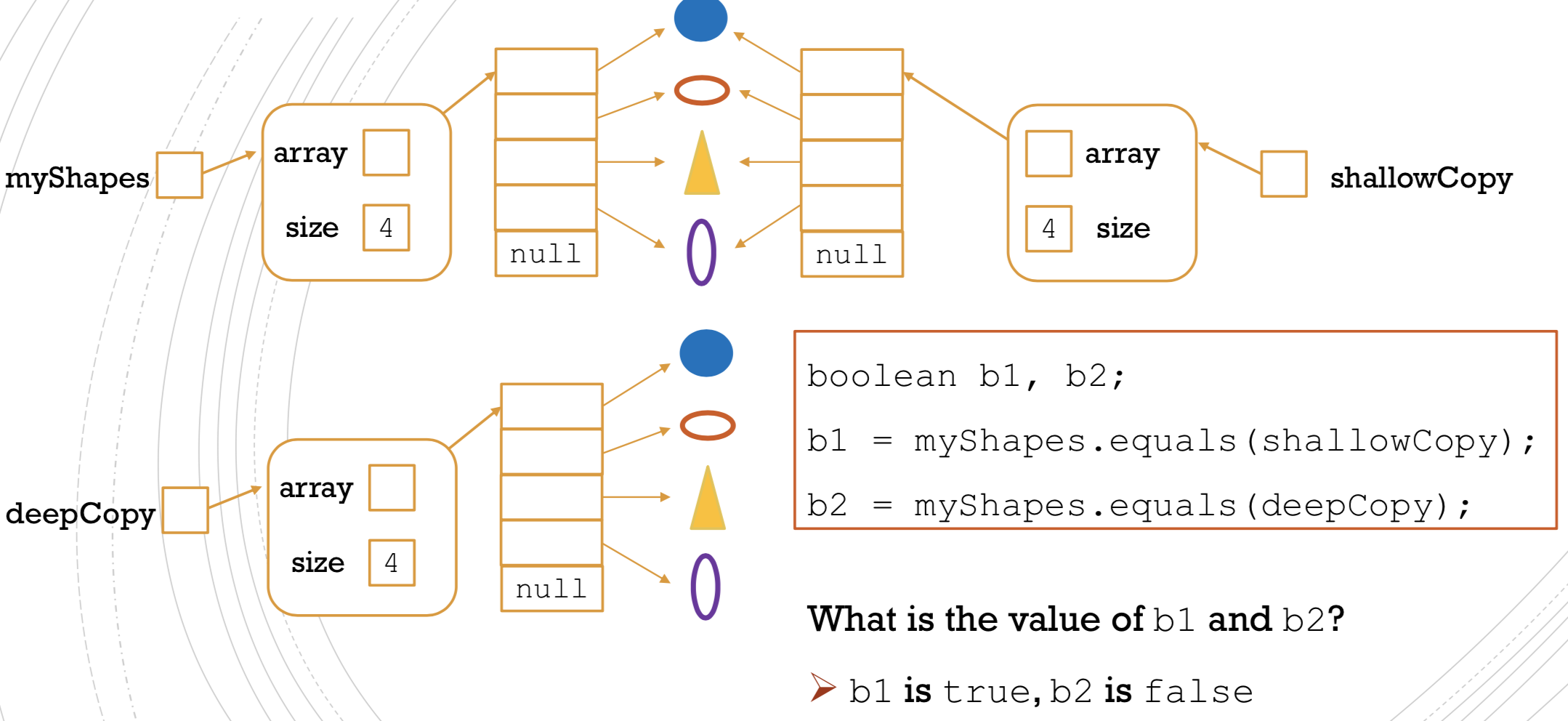
String overrides

* clone()

*Creates and returns a copy of this Object.*

* x.clone()and x points to objects of the same type.
* x.clone() == x is false
* x.clone().equals(x) is true

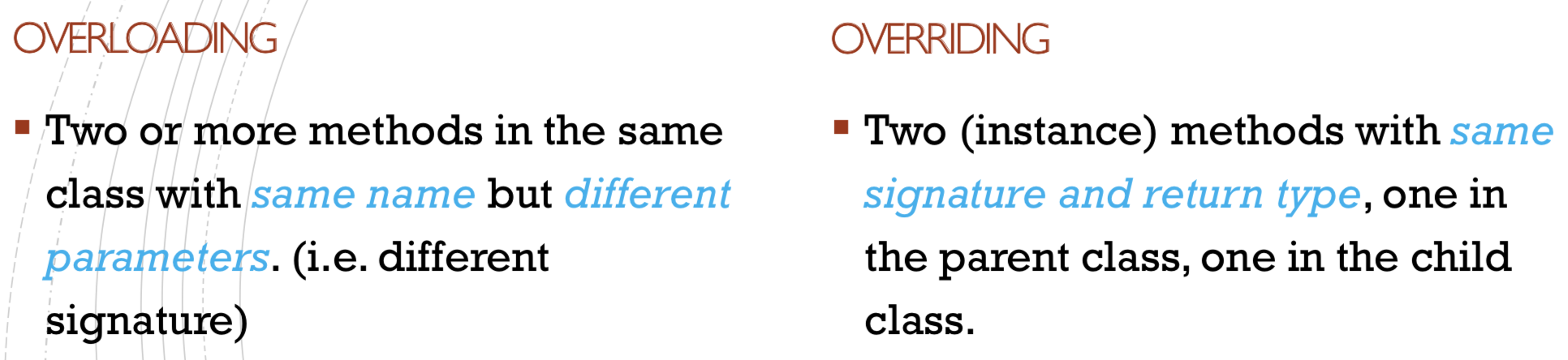
By convention, when overriding clone, super.clone() should be called

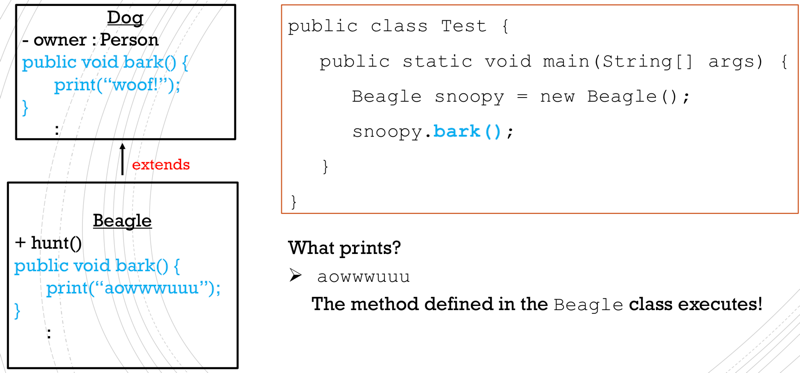


Subclass

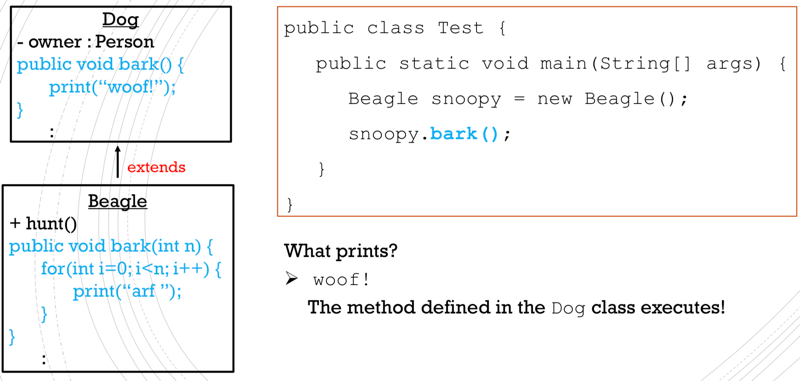
* Fields
* The inherited fields can be used as any other field.
* If you declare a field with the same name as the one in the superclass in the subclass, you **hide** the inherited attribute. (you should NOT do this)
* You can declare new fields.
* Methods
* The inherited methods can be used as they are.
* If you write a non-static method with the same signature (and same return type) as the one from the superclass, you are **overriding** the method.
* If you write a static method with the same signature (and same return type) as the one from the superclass, you are **hiding** the method.
* You can declare new methods in the subclass.

overloading VS overriding





overloading



overriding

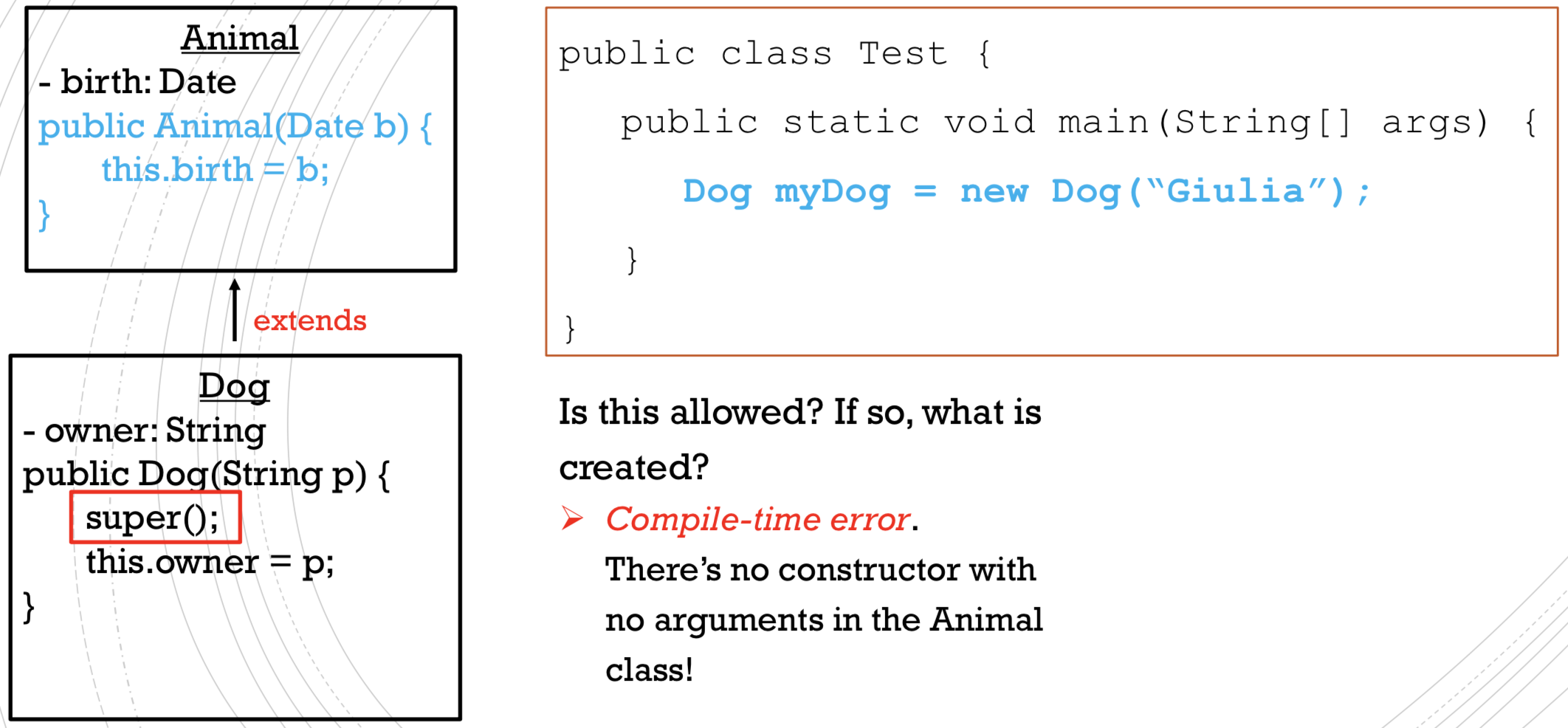
Keyword: super

* *Super refers to such object as an instance of the superclass and we can use super to access attributes and methods of the superclass.*

In general, it is not needed (since the subclass inherits all members of the superclass). It must be used if the method you want to access has been **overridden** or if the field has been **hidden**.

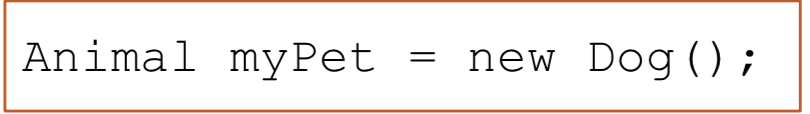
* *Inside the subclass constructors to invoke a constructor from the superclass*



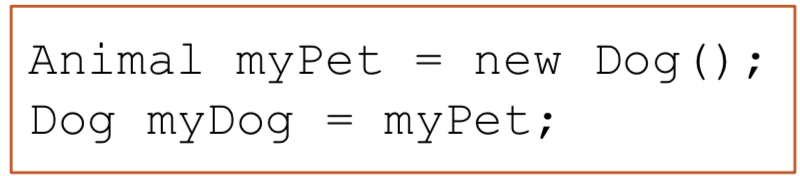


Type Conversion

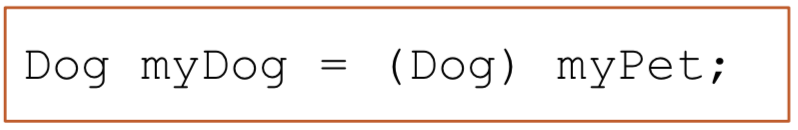
*Casting allows us to use an object of one type in place of another type, if permitted.*



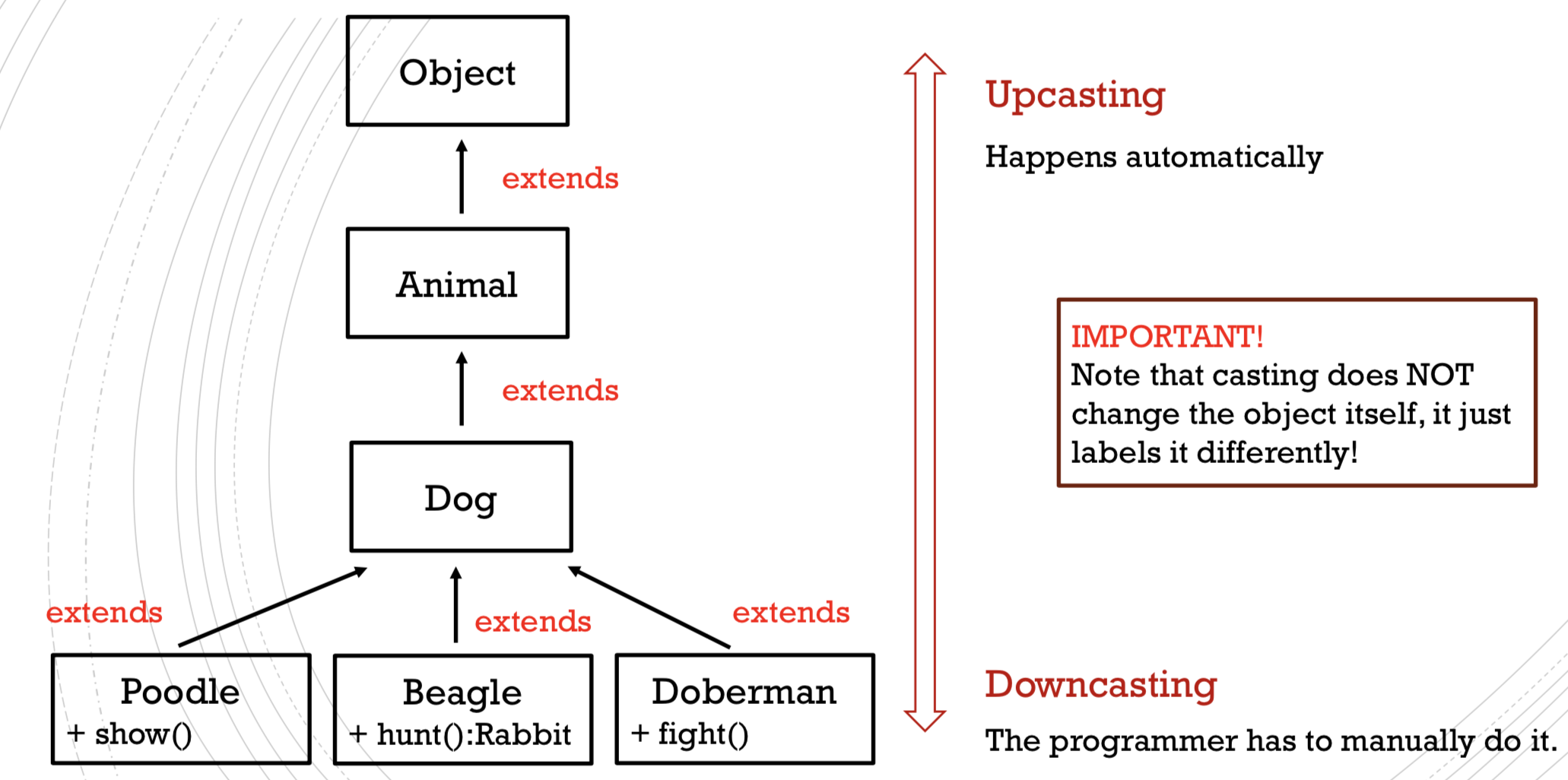
This will not cause a compile-time error because there is an **implicit upcasting** since a Dog is for sure also an Animal.



The second line will cause a compile-time error. From the compiler point of view, myPet is of type Animal and an Animal might not be a Dog. However, we can tell the compiler that myPet is of the correct type, by **explicitly downcasting**



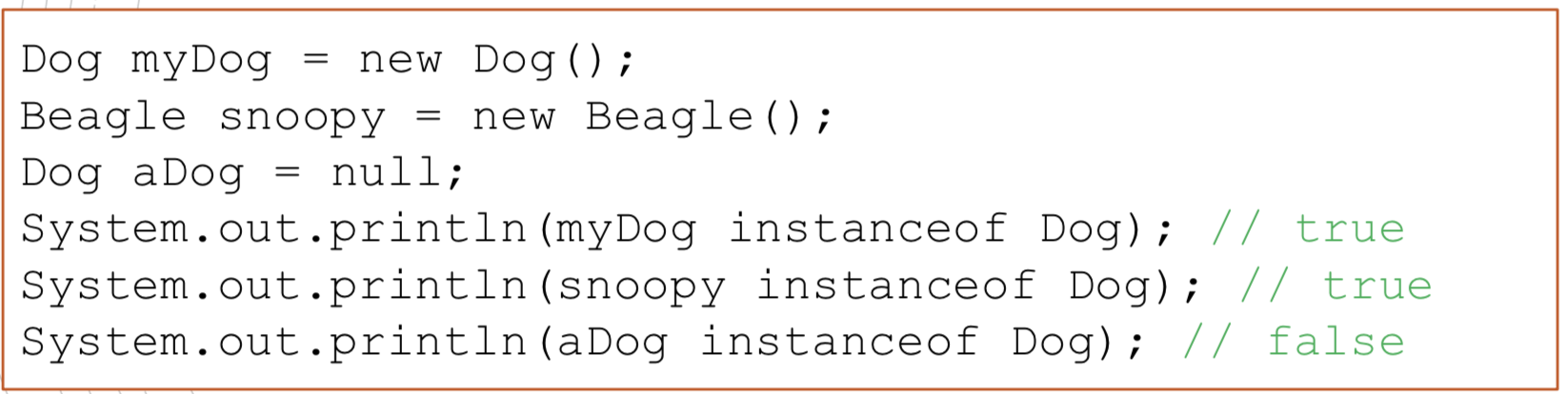
If myPet turns out to be of the wrong type we’ll get a run-time error.



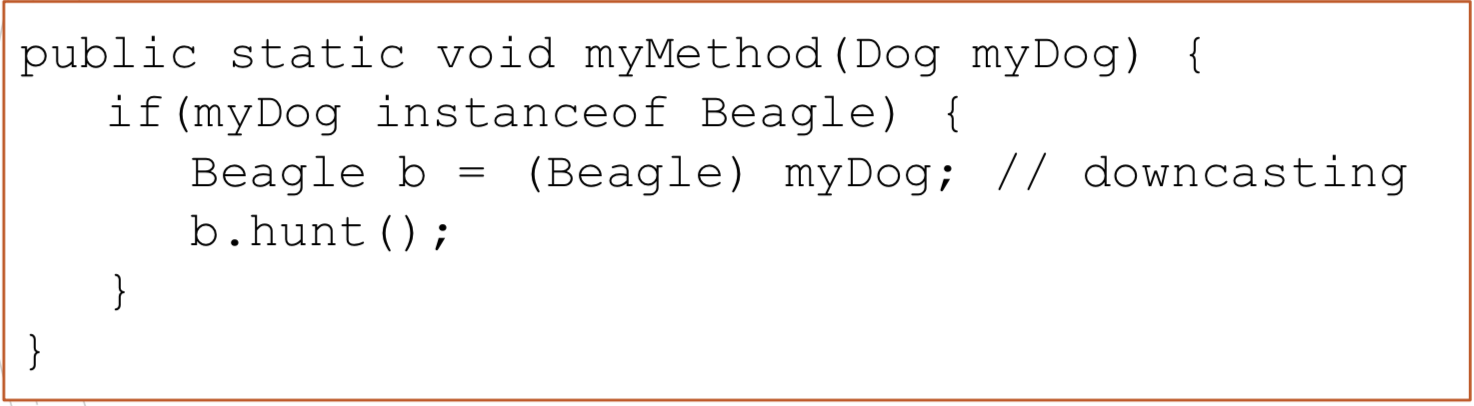
* instanceof

*The instanceof operator is used to test whether an object is an instance of the specified type.*

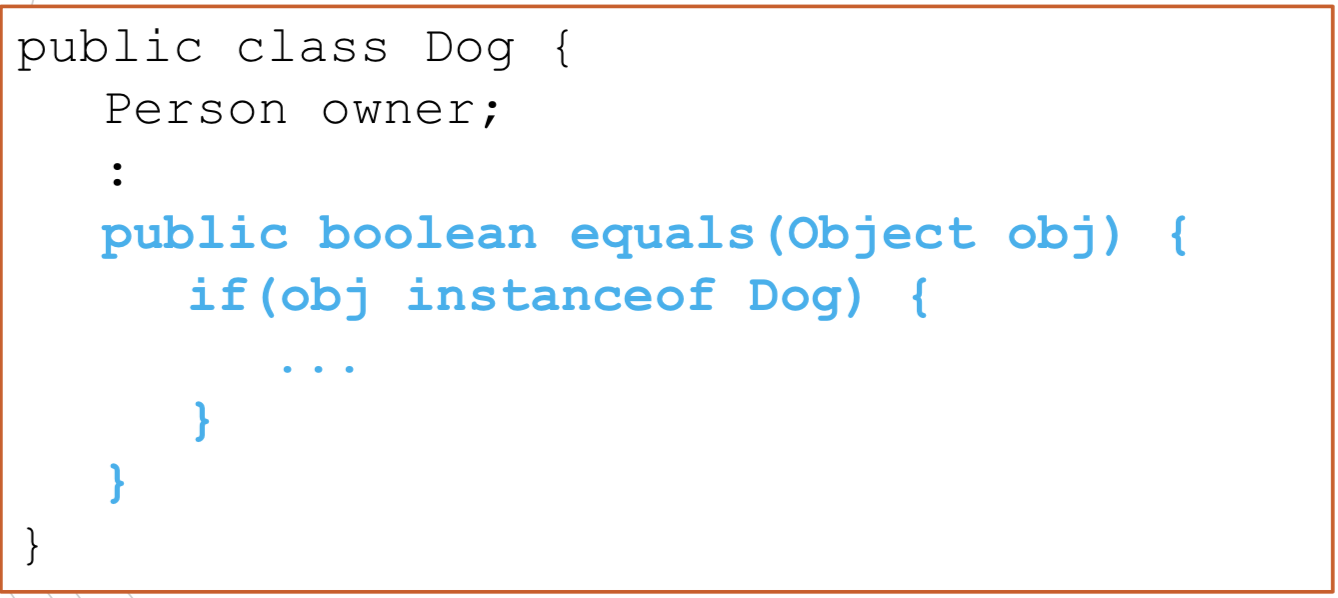
It returns either true or false. If we apply the instanceof operator with any variable that has **null** value, it returns false.



Use instanceof to make sure that **downcasting** to a subclass will not cause a run-time error.



We have to use instanceof when **overriding equals()**



Polymorphism

Each object can have different “forms”.

One important aspect of polymorphism in Java: “the Java virtual machine (JVM) calls the appropriate method for the object that is referred to in each variable. It does not call the method that is defined by the variable's type”.

