COMP 250
INTRODUCTION TO COMPUTER SCIENCE

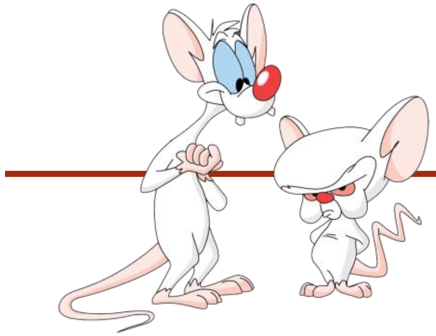
Lecture 16 – Comparable and Iterable

Giulia Alberini, Fall 2018

FROM LAST CLASS

- Comparable
- Iterable
- Iterator

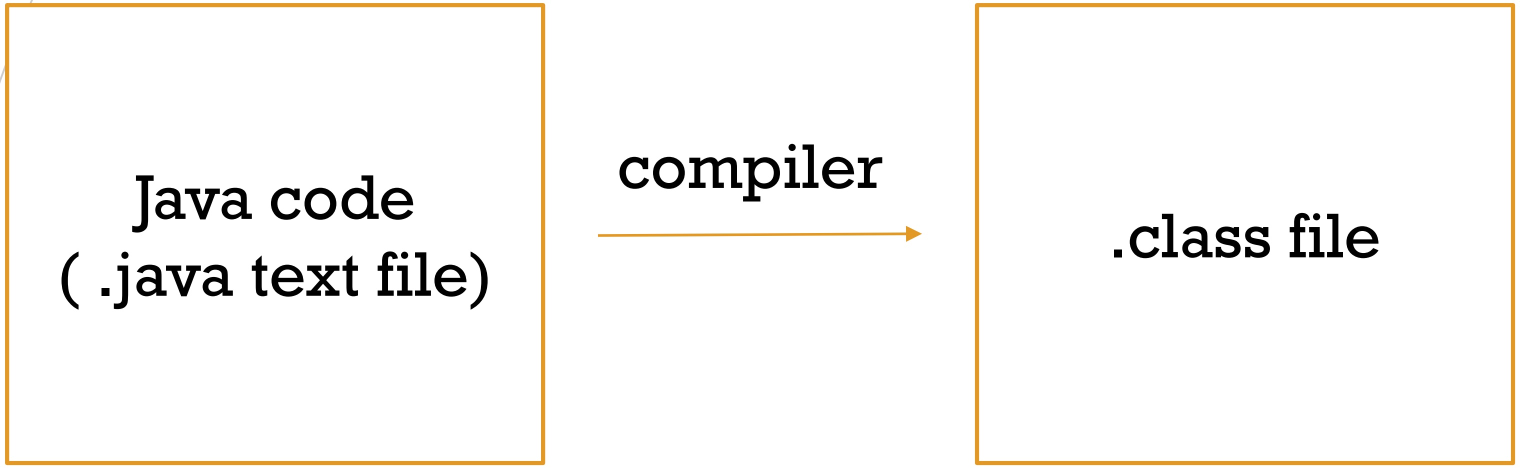
WHAT ARE WE GOING TO DO TODAY?



- class Class
- Memory allocation

Class

.java AND .class FILES



Java code
(.java text file)

compiler

.class file

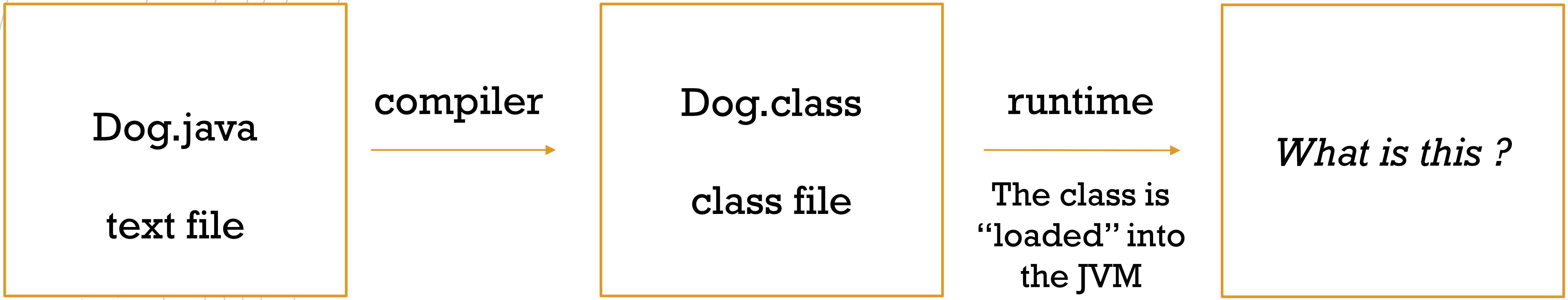
JAVA .class FILE ("BYTE CODE")

It has a specific format for information such as:

- the class name
- fields (names, types)
- methods (signature, return type, instructions)
- superclass
- ….

https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-4.html

EXAMPLE



Dog.java

text file

compiler

Dog.class

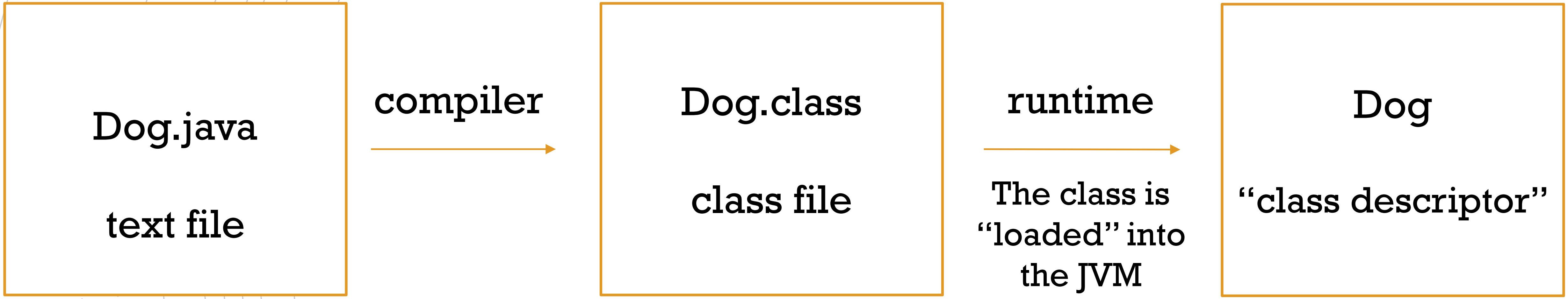
class file

runtime

The class is "loaded" into the JVM

What is this ?

EXAMPLE



Dog.java

text file

compiler

Dog.class

class file

runtime

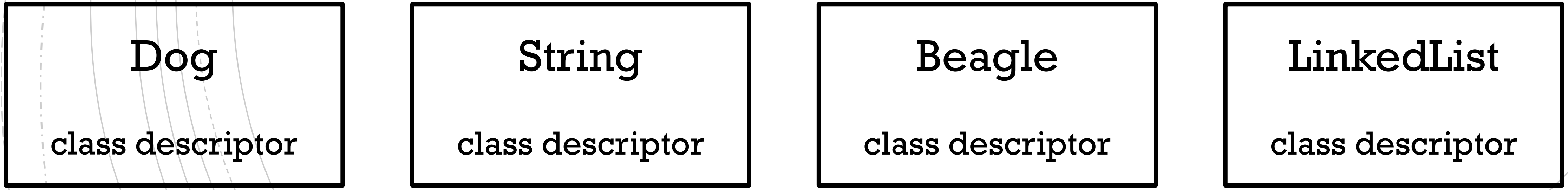
The class is “loaded” into the JVM

Dog

“class descriptor”

"CLASS DESCRIPTORS"

- The term "class descriptor" is not standard. So don't look it up.
- It is an *object* that contains all the information about a class.
- If it is an object, then what class is it an instance of?



Dog

class descriptor

String

class descriptor

Beagle

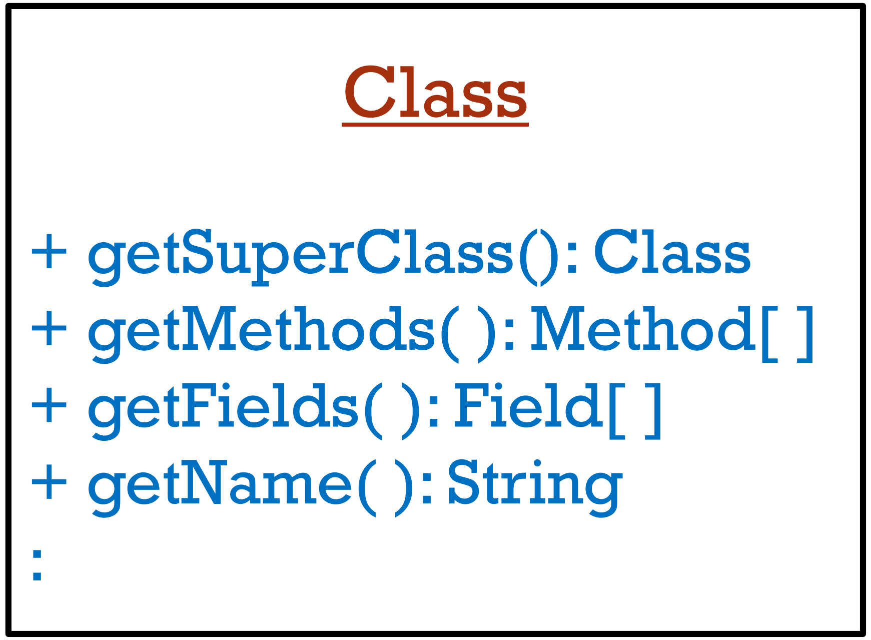
class descriptor

LinkedList

class descriptor

`Class`

- The class `Class` is part of the `java.lang` package.
- A “class descriptor” is an instance of the class `Class`.
- Instances of the class `Class` represent classes and interfaces in a running Java application.



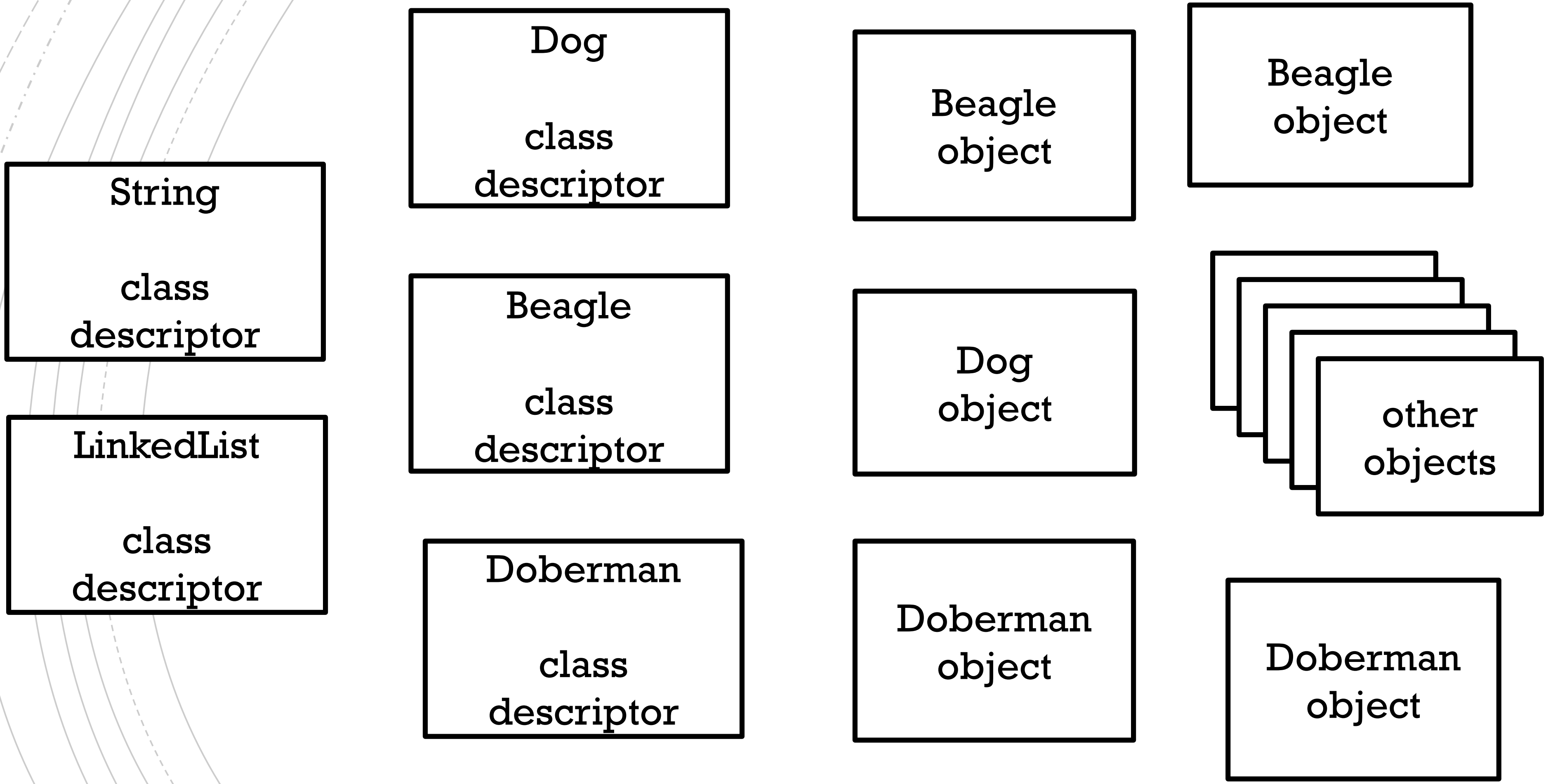
Class

+ getSuperClass(): Class
+ getMethods( ): Method[ ]
+ getFields( ): Field[ ]
+ getName( ): String
:

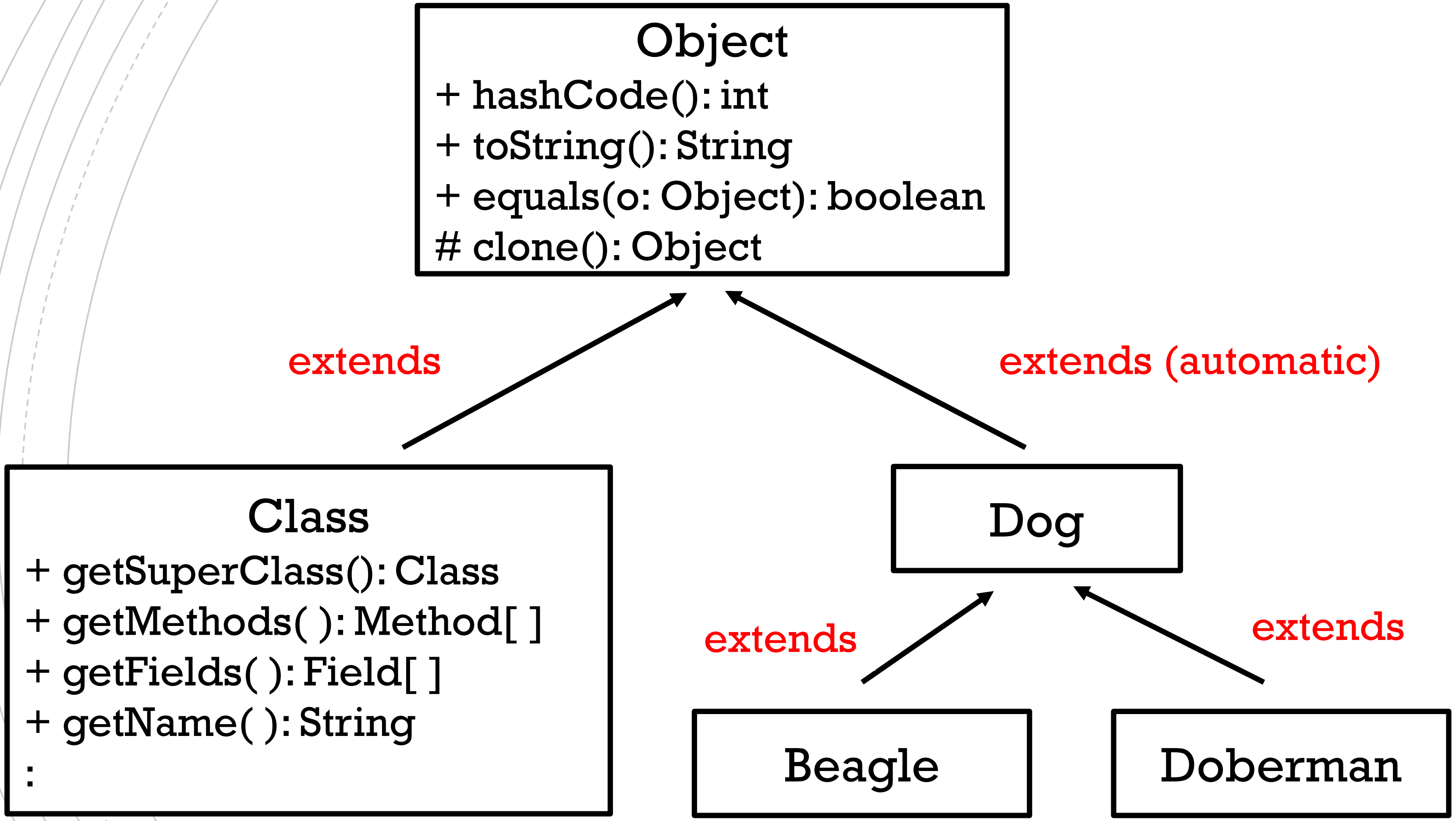
INSTANCES OF CLASSES

- **A** `Dog` **object is an instance of the** `Dog` **class.**
- **A** `String` **object is an instance of the** `String` **class.**
- **An** `Object` **object is an instance of the** `Object` **class.**
- ***A*** *`Class`* ***object (“class descriptor” object) is an instance of the*** *`Class`* ***class.***

EXAMPLE OF OBJECTS IN A RUNNING JAVA PROGRAM



CLASS HIERARCHY IN THE EXAMPLE



Object
+ hashCode(): int
+ toString(): String
+ equals(o: Object): boolean
# clone(): Object

extends

extends (automatic)

Class
+ getSuperClass(): Class
+ getMethods( ): Method[ ]
+ getFields( ): Field[ ]
+ getName( ): String
:

Dog

extends

extends

Beagle

Doberman

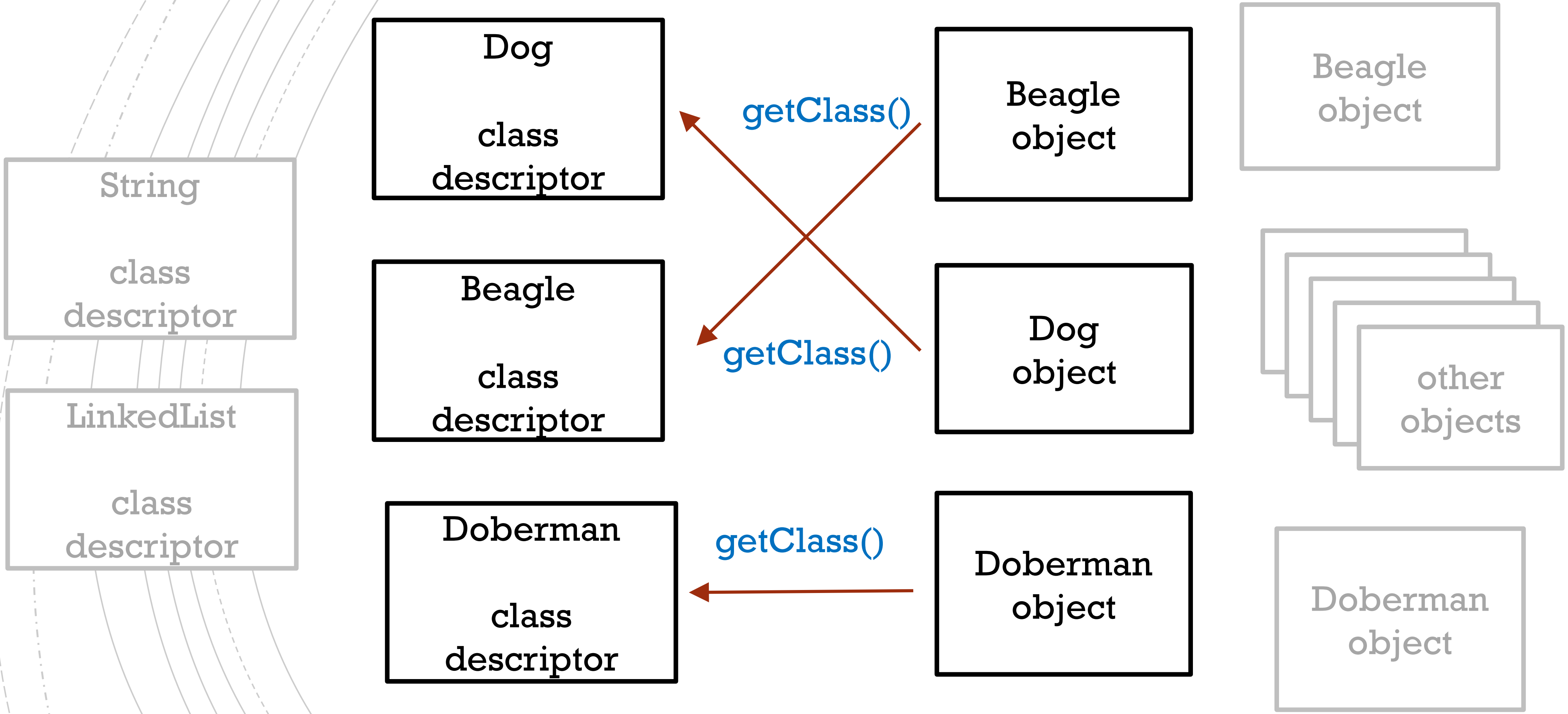
getClass()

- All classes inherit a method called `getClass()` from the `Object` class.
- This method returns the run-time class of *this* object

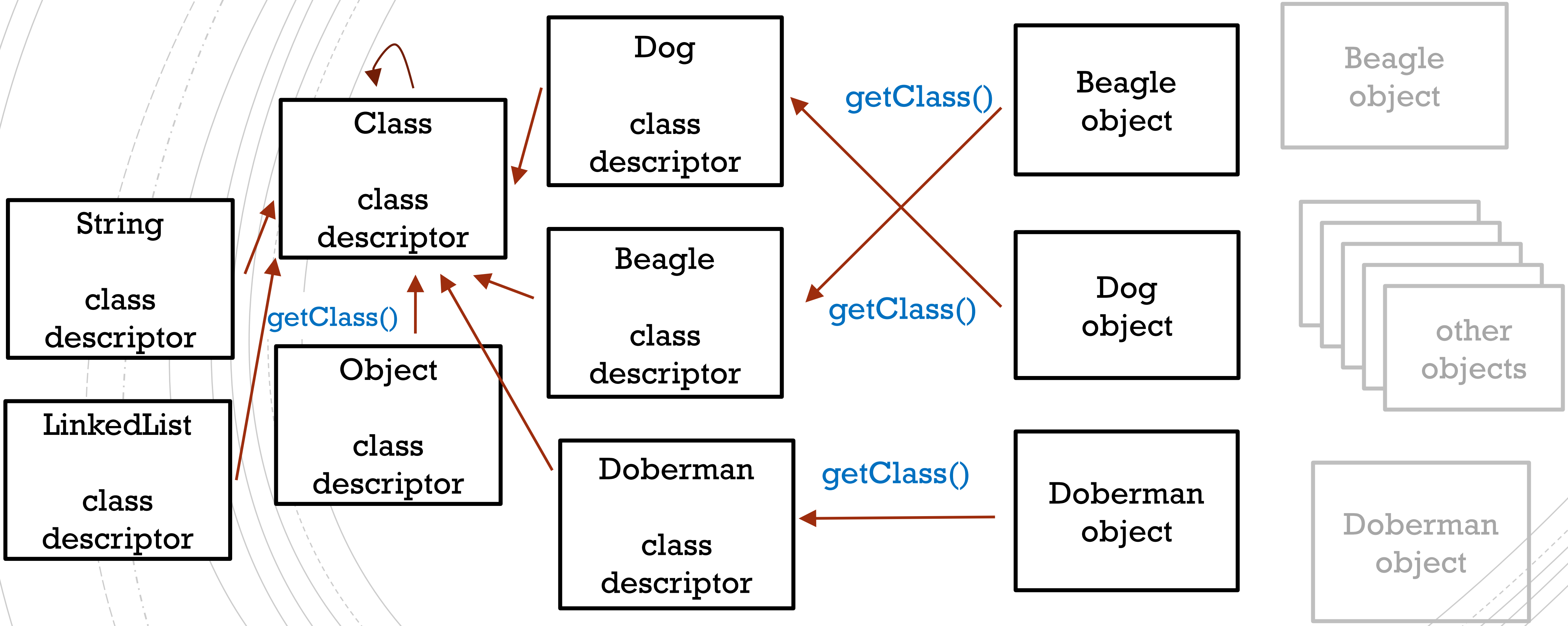
```
AnyClass a = new AnyClass();

Class c = a.getClass();
```

EXAMPLE OF OBJECTS IN A RUNNING JAVA PROGRAM



EXAMPLE OF OBJECTS IN A RUNNING JAVA PROGRAM

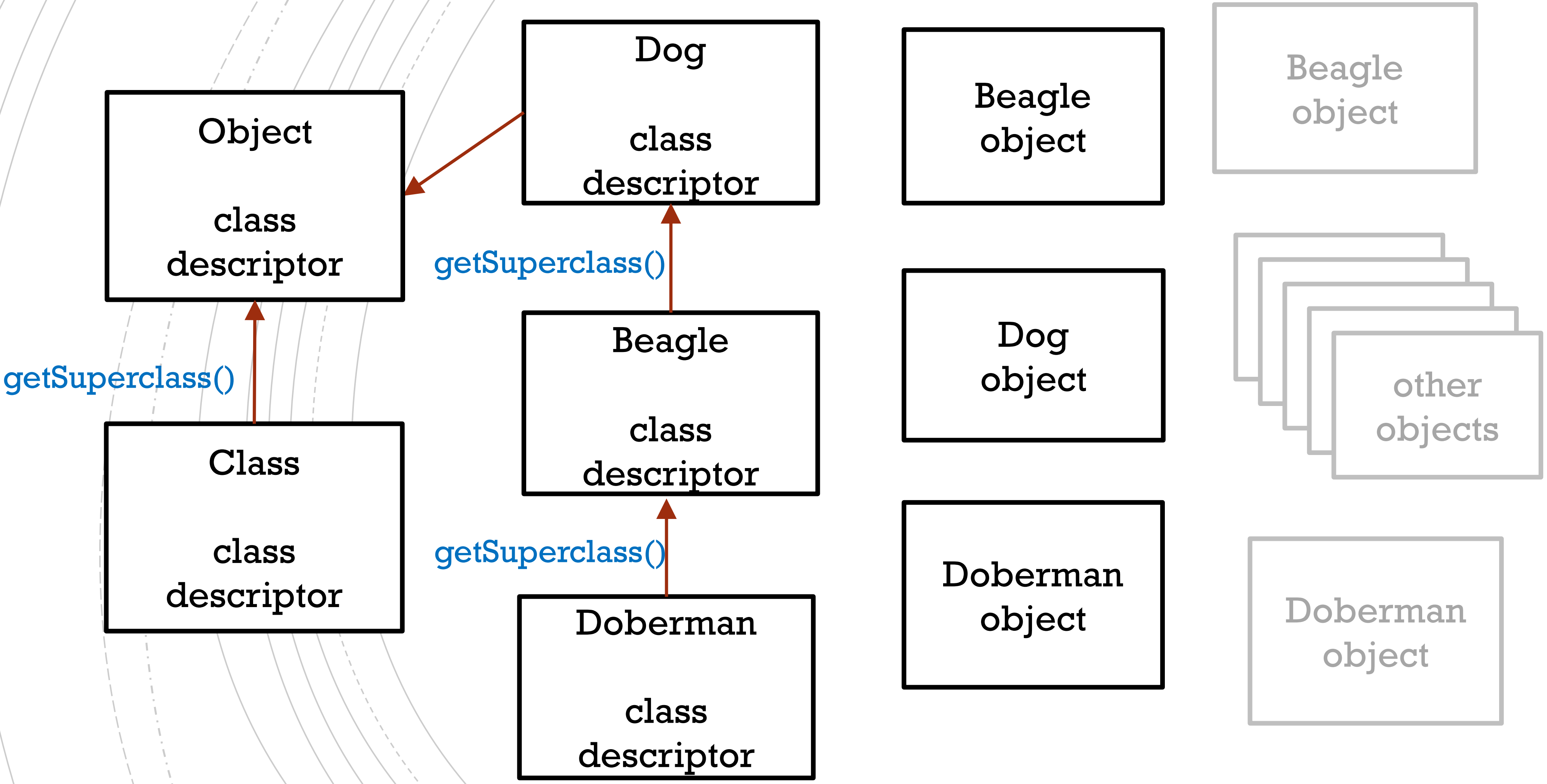


`getSuperclass()`

- This is one of the methods from the class `Class`.

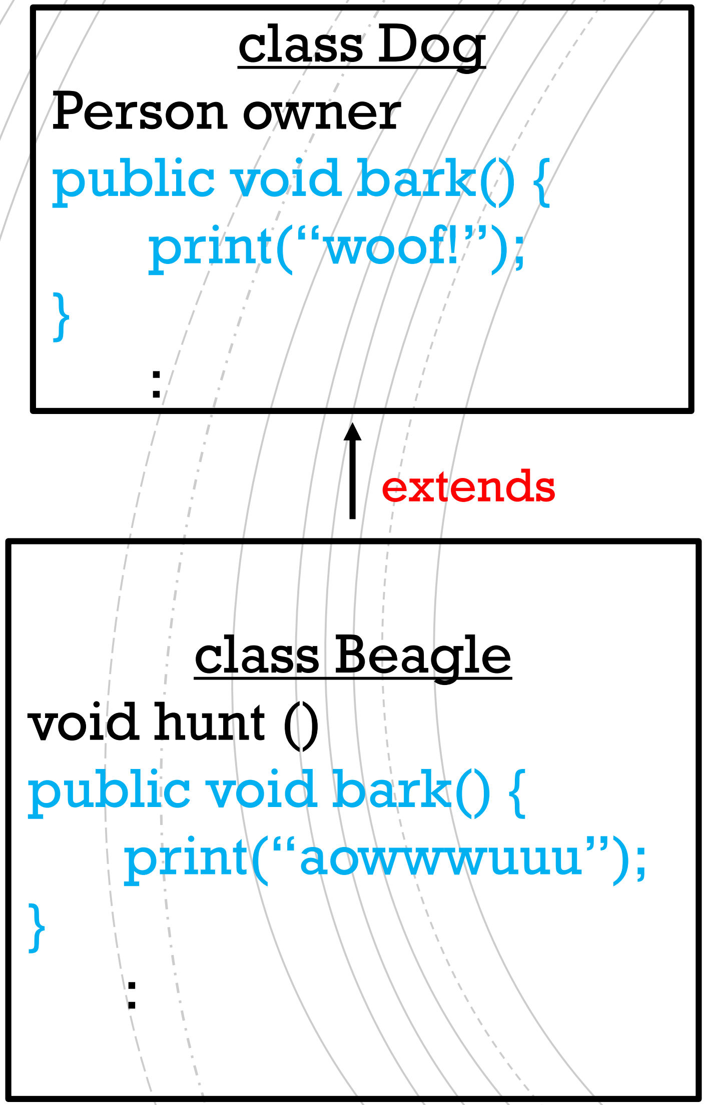
- It returns the `Class` representing the superclass of the class represented by *this* `Class`.

EXAMPLE



The method cannot be invoked by the other objects in the picture, why?

REMEMBER WHEN WE TALKED ABOUT POLYMORPHISM?



class Dog

```
Person owner
public void bark() {
      print(“woof!”);
}
      :
```

extends

class Beagle

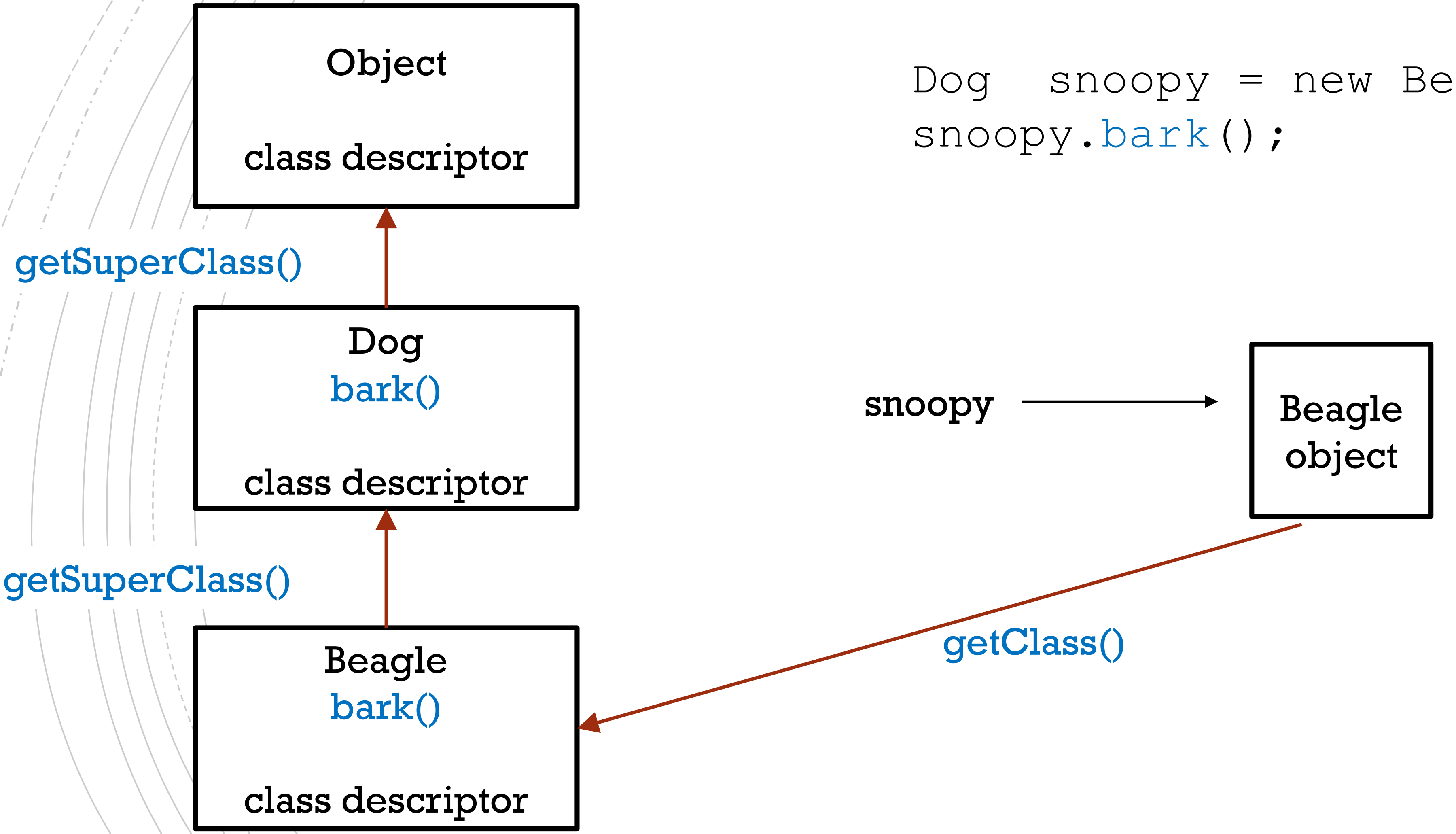
```
void hunt ()
public void bark() {
      print(“aowwwuuu”);
}
      :
```

```
public class Test {
   public static void main(String[] args) {
      Dog snoopy = new Beagle();
      snoopy.bark();
   }
}
```

Which `bark()` will execute???

WHEN THE PROGRAM IS RUNNING…

```
Dog  snoopy = new Beagle();
snoopy.bark();
```



MEMORY ALLOCATION – HEAP VS STACK

- The Java Virtual Machine (JVM) divides memory between Java Heap Space and Java Stack Memory

- Java Heap space is used by java runtime to allocate memory to Objects and JRE classes. Whenever we create any object, it's always created in the Heap space.

- Java Stack memory is used for execution of a thread. They contain method specific values and references to other objects in the heap that are getting referred from the method.

JAVA STACK

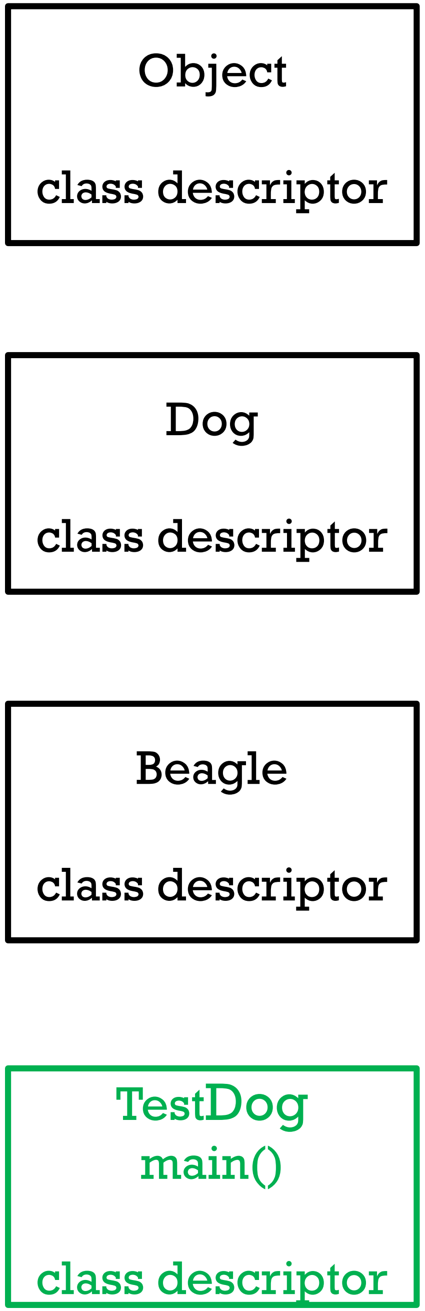
Java stack memory uses a LIFO data structure.

- Each time a method is invoked, it creates a new block in the stack for that particular method.
- Each method block has all the local values, as well as references to other objects that are being used by the method.
- When the method ends, its block will be erased and will be available for use by the next method.
- The values stored in each block are accessible only from that particular method.

JAVA HEAP SPACE

- Whenever we create any object, it’s always created in the Heap space.
- There is no specific order in reserving blocks in a heap.
- Any object created in the heap space has global access and can be referenced from anywhere of the application.
- Garbage Collection runs on the heap memory to free the memory used by objects that doesn’t have any reference.

Suppose we are running a class TestDog, which has a main() method.



Object

class descriptor

Dog

class descriptor

Beagle

class descriptor

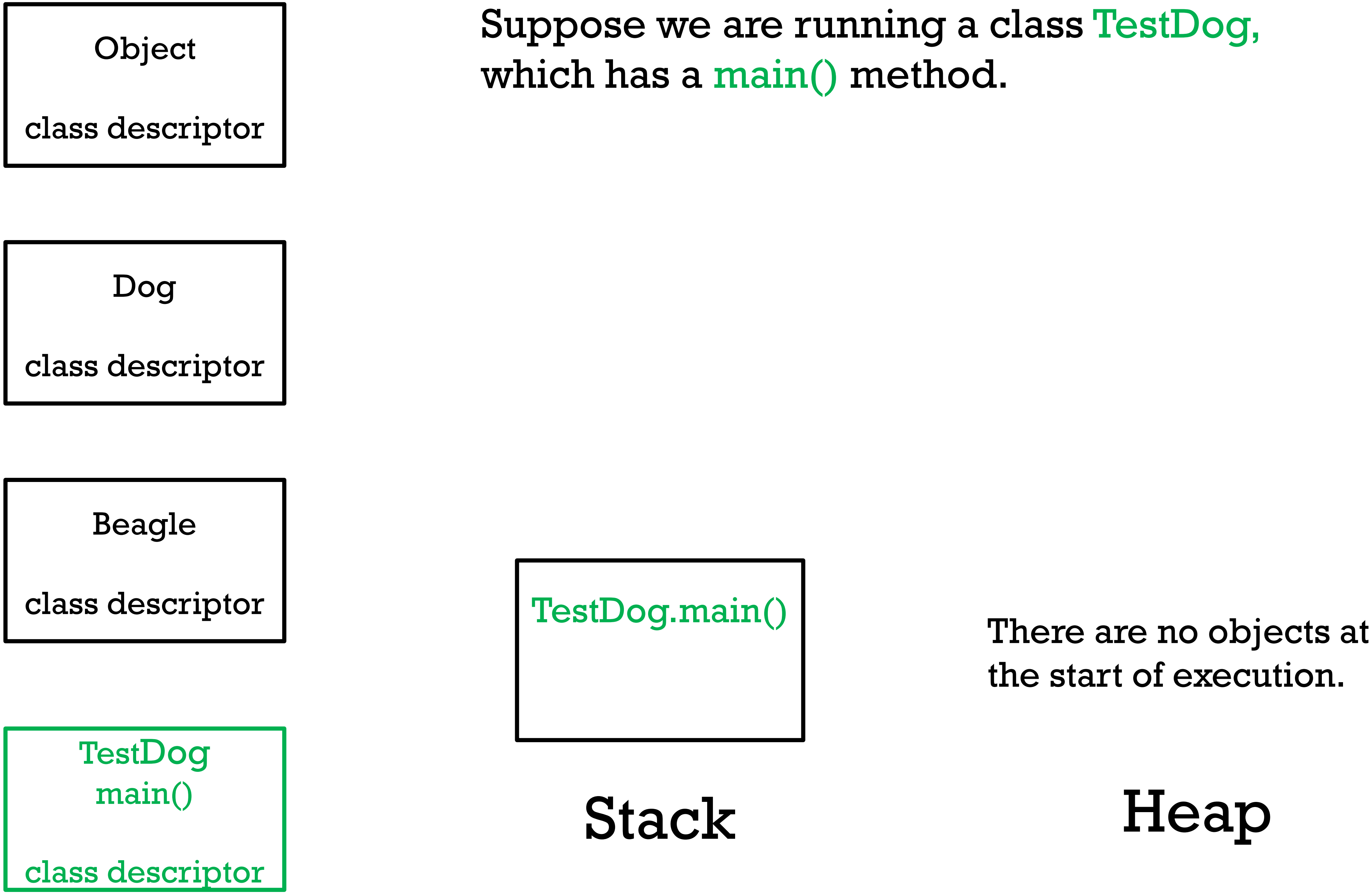
TestDog
main()

class descriptor

Permanent Generation (non-heap): The pool containing all the reflective data of the virtual machine itself, such as class and method objects.

https://docs.oracle.com/javase/7/docs/technotes/guides/management/jconsole.html

Suppose we are running a class TestDog, which has a main() method.



Object

class descriptor

Dog

class descriptor

Beagle

class descriptor

TestDog
main()

class descriptor

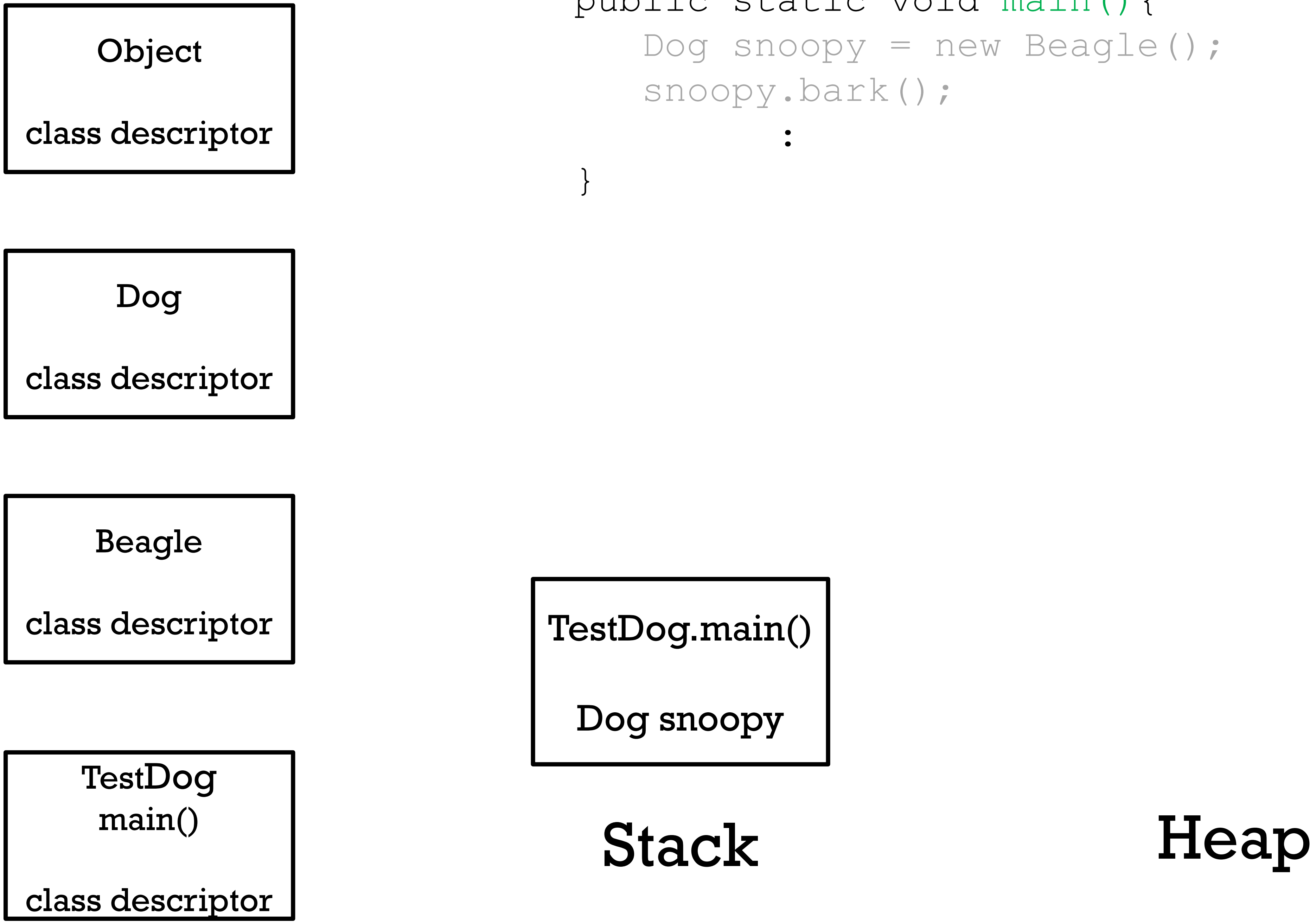
TestDog.main()

Stack

There are no objects at the start of execution.

Heap

```
public static void main(){
    Dog snoopy = new Beagle();
    snoopy.bark();
              :
}
```



Object

class descriptor

Dog

class descriptor

Beagle

class descriptor

TestDog
main()

class descriptor

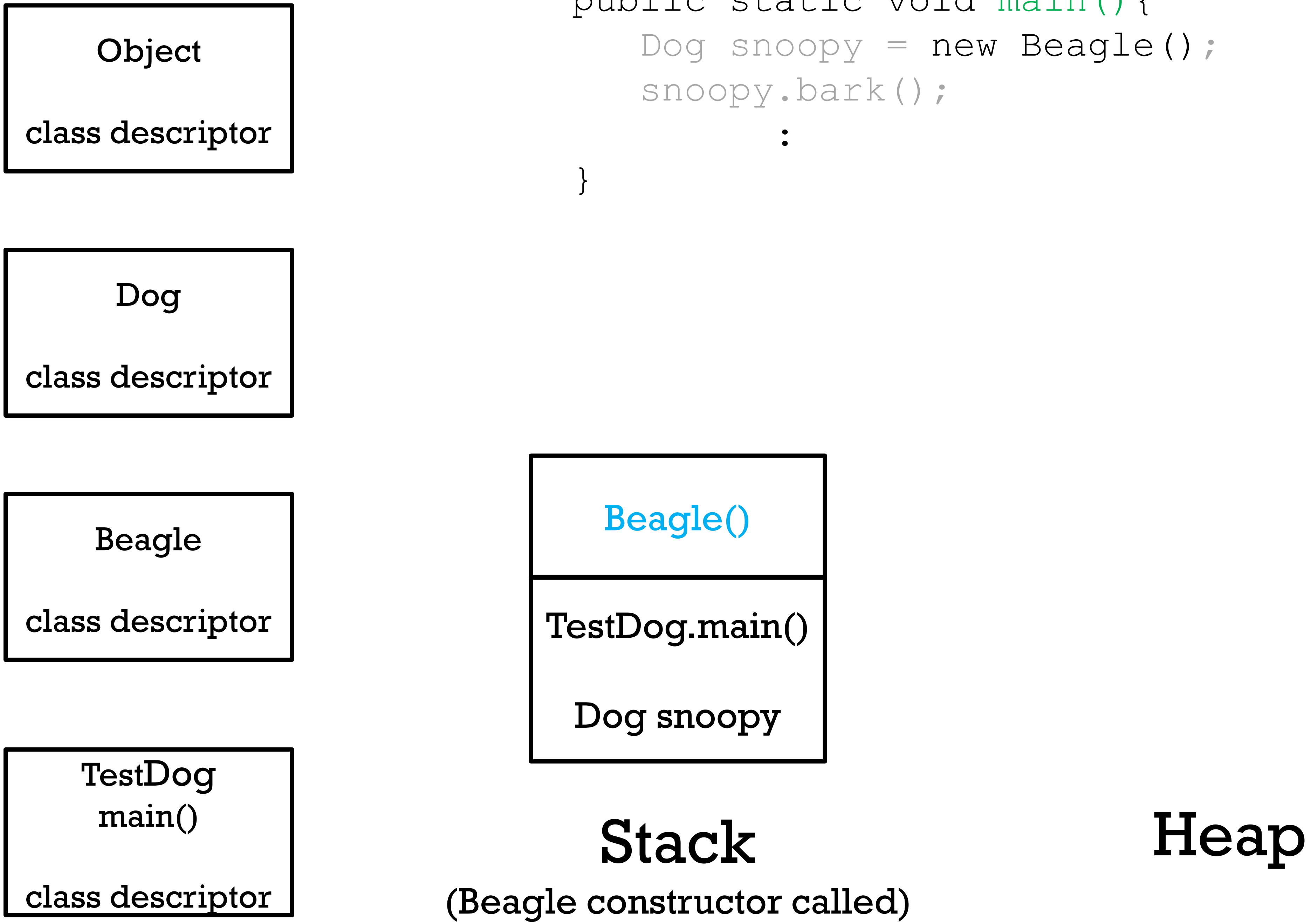
TestDog.main()

Dog snoopy

Stack

Heap

```
public static void main(){
    Dog snoopy = new Beagle();
    snoopy.bark();
            :
}
```



Object

class descriptor

Dog

class descriptor

Beagle

class descriptor

TestDog
main()

class descriptor

Beagle()

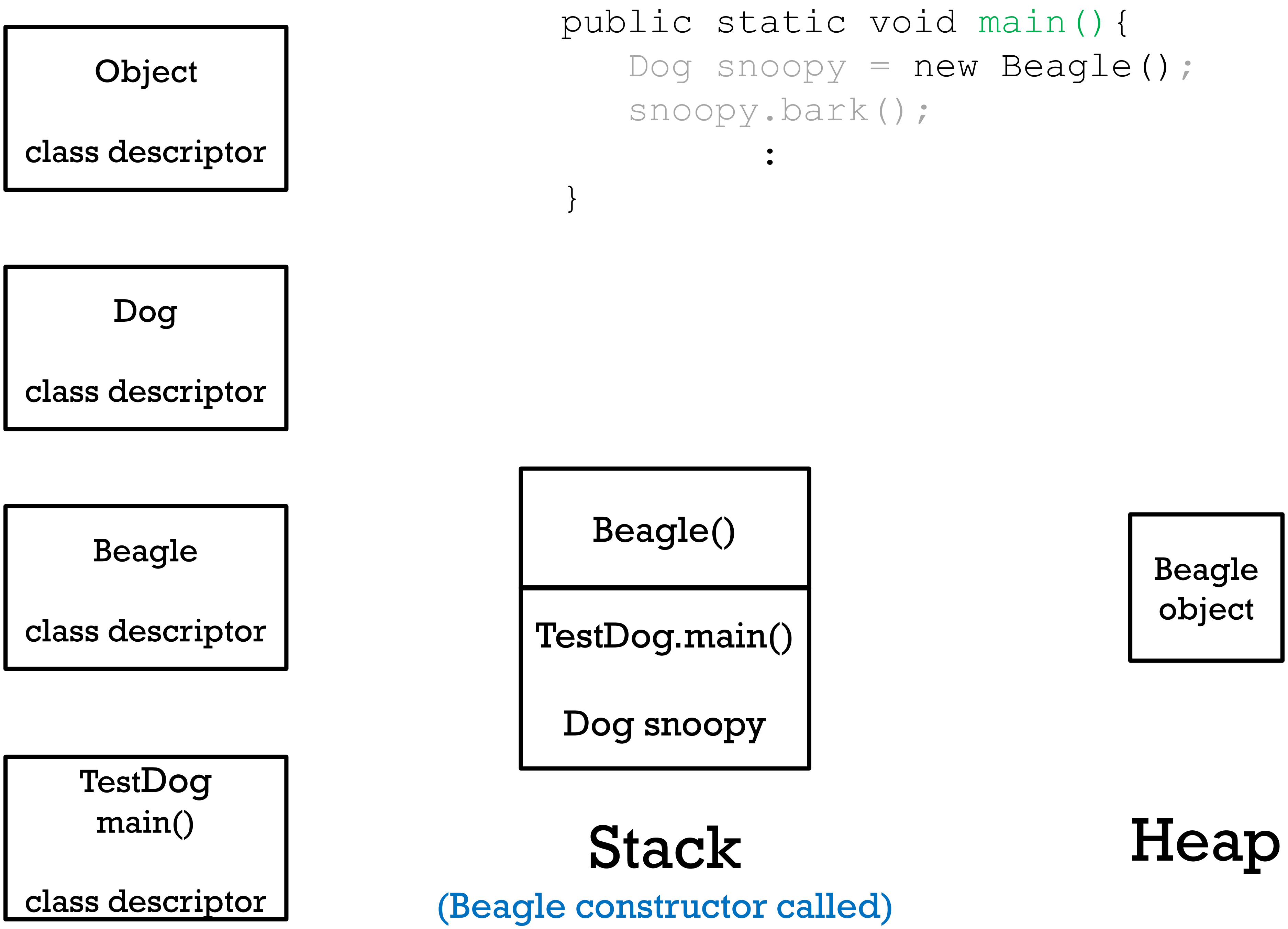
TestDog.main()

Dog snoopy

Stack

(Beagle constructor called)

Heap



```
public static void main(){
    Dog snoopy = new Beagle();
    snoopy.bark();
                 :
}
```

Object

class descriptor

Dog

class descriptor

Beagle

class descriptor

TestDog
main()

class descriptor

Beagle()

TestDog.main()

Dog snoopy

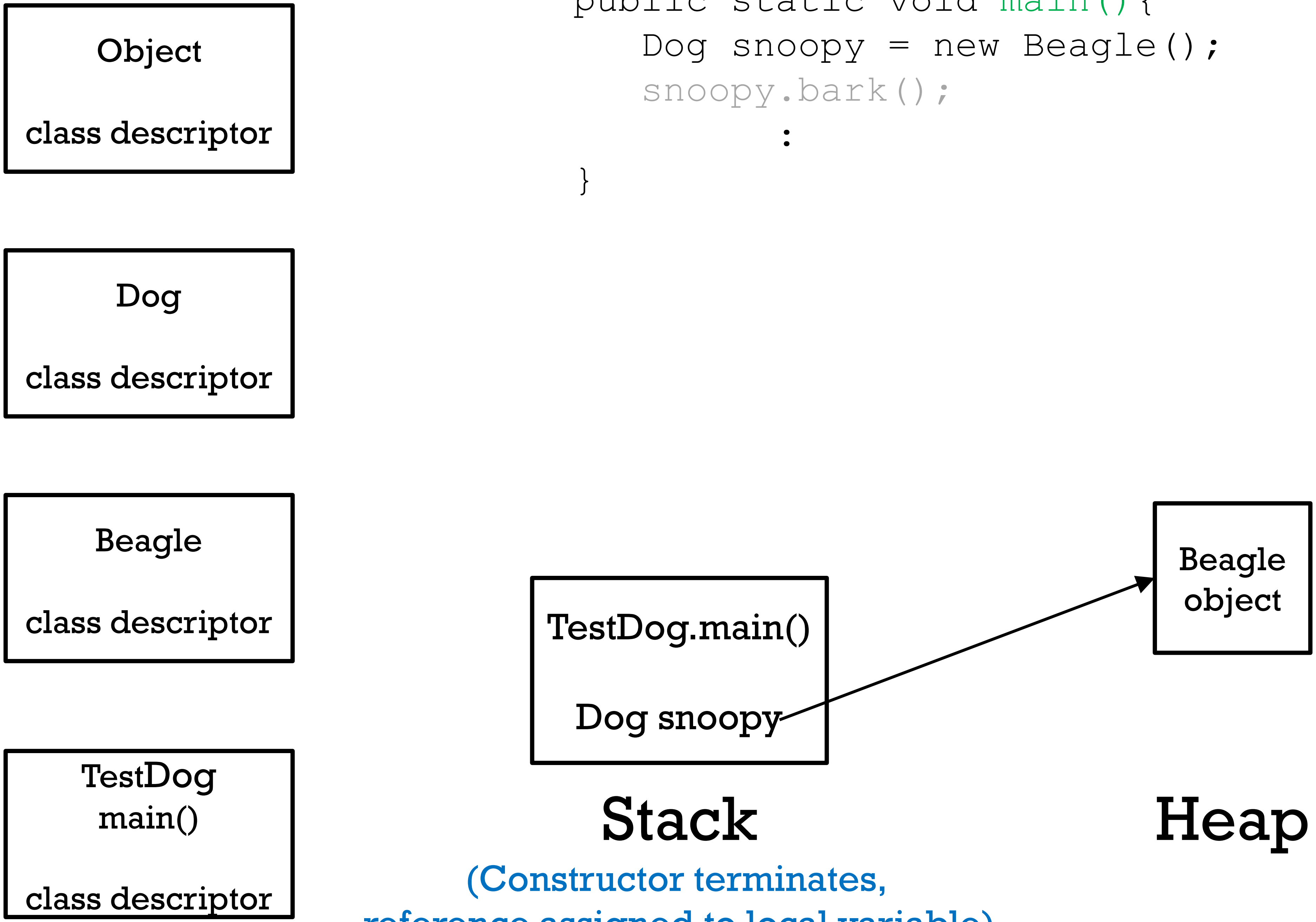
Beagle object

Stack

(Beagle constructor called)

Heap

```
public static void main(){
    Dog snoopy = new Beagle();
    snoopy.bark();
              :
}
```



Object

class descriptor

Dog

class descriptor

Beagle

class descriptor

TestDog
main()

class descriptor

TestDog.main()

Dog snoopy

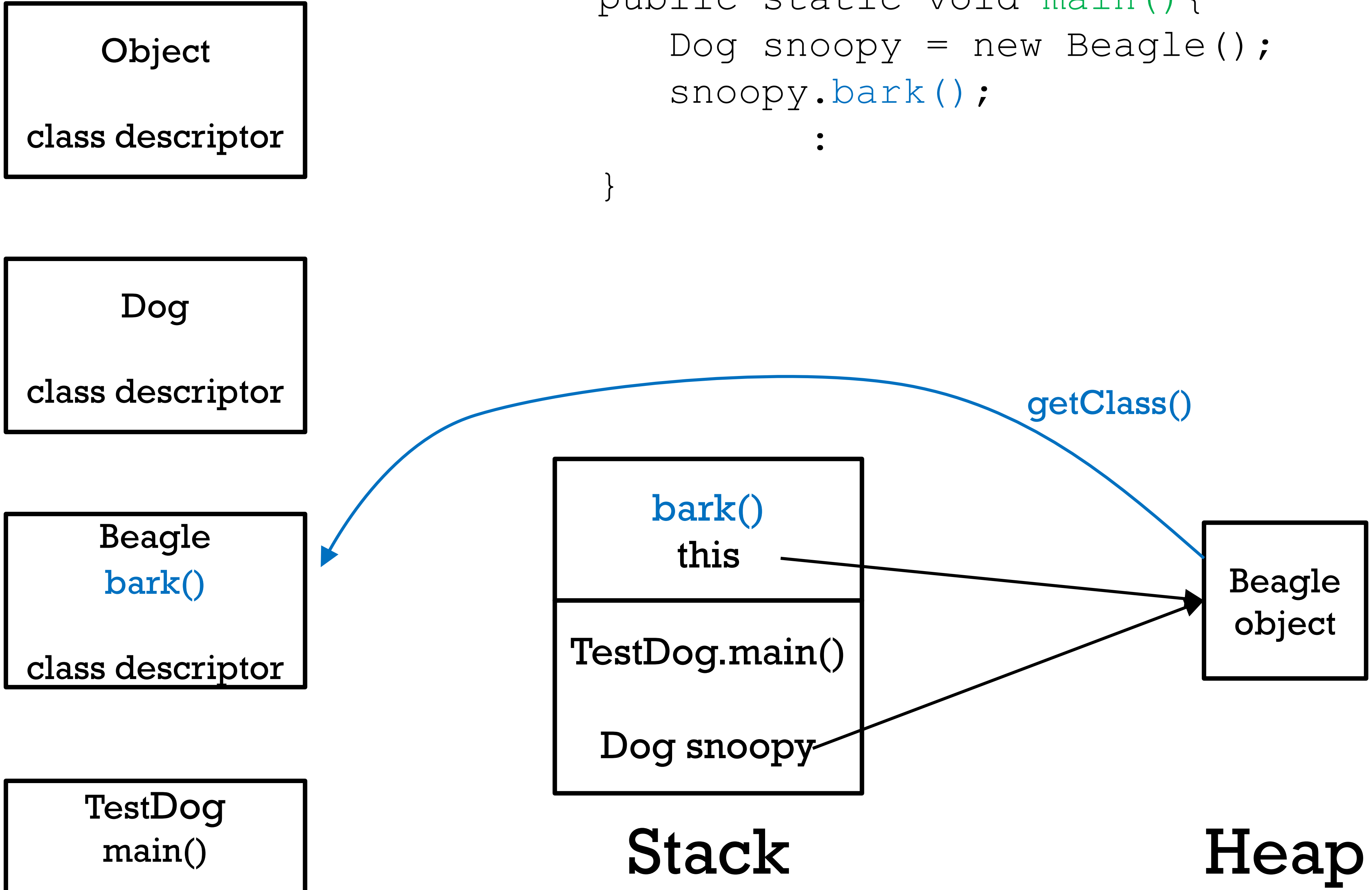
Beagle object

Stack

Heap

(Constructor terminates,
reference assigned to local variable)

```
public static void main(){
    Dog snoopy = new Beagle();
    snoopy.bark();
            :
}
```



Object

class descriptor

Dog

class descriptor

Beagle

bark()

class descriptor

TestDog

main()

class descriptor

getClass()

bark()

this

TestDog.main()

Dog snoopy

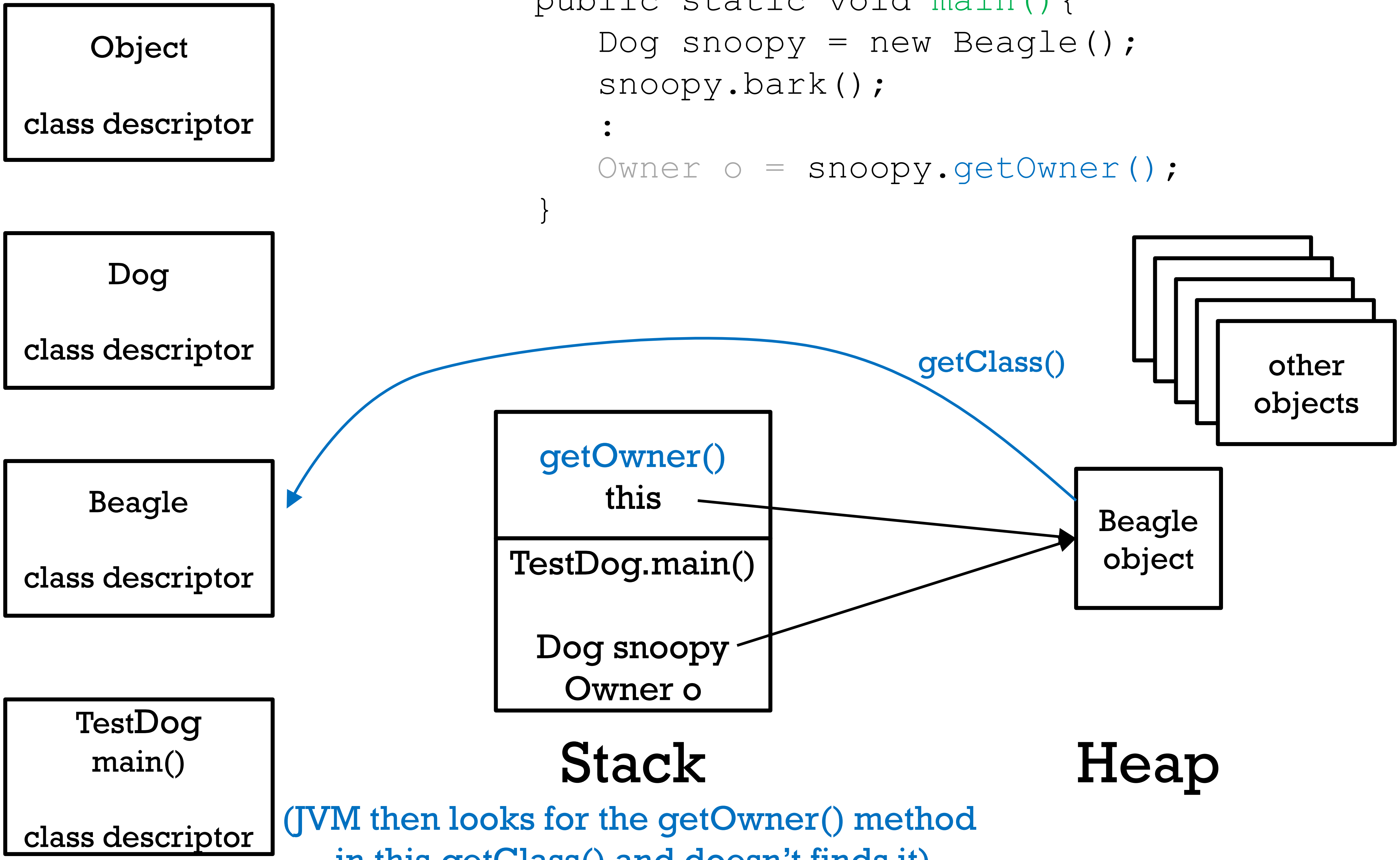
Beagle object

Stack

Heap

(JVM looks for the bark() method in this.getClass() and finds it)

```
public static void main(){
    Dog snoopy = new Beagle();
    snoopy.bark();
    :
    Owner o = snoopy.getOwner();
}
```



Object

class descriptor

Dog

class descriptor

Beagle

class descriptor

TestDog
main()

class descriptor

getClass()

other objects

getOwner()
this

TestDog.main()

Dog snoopy
Owner o

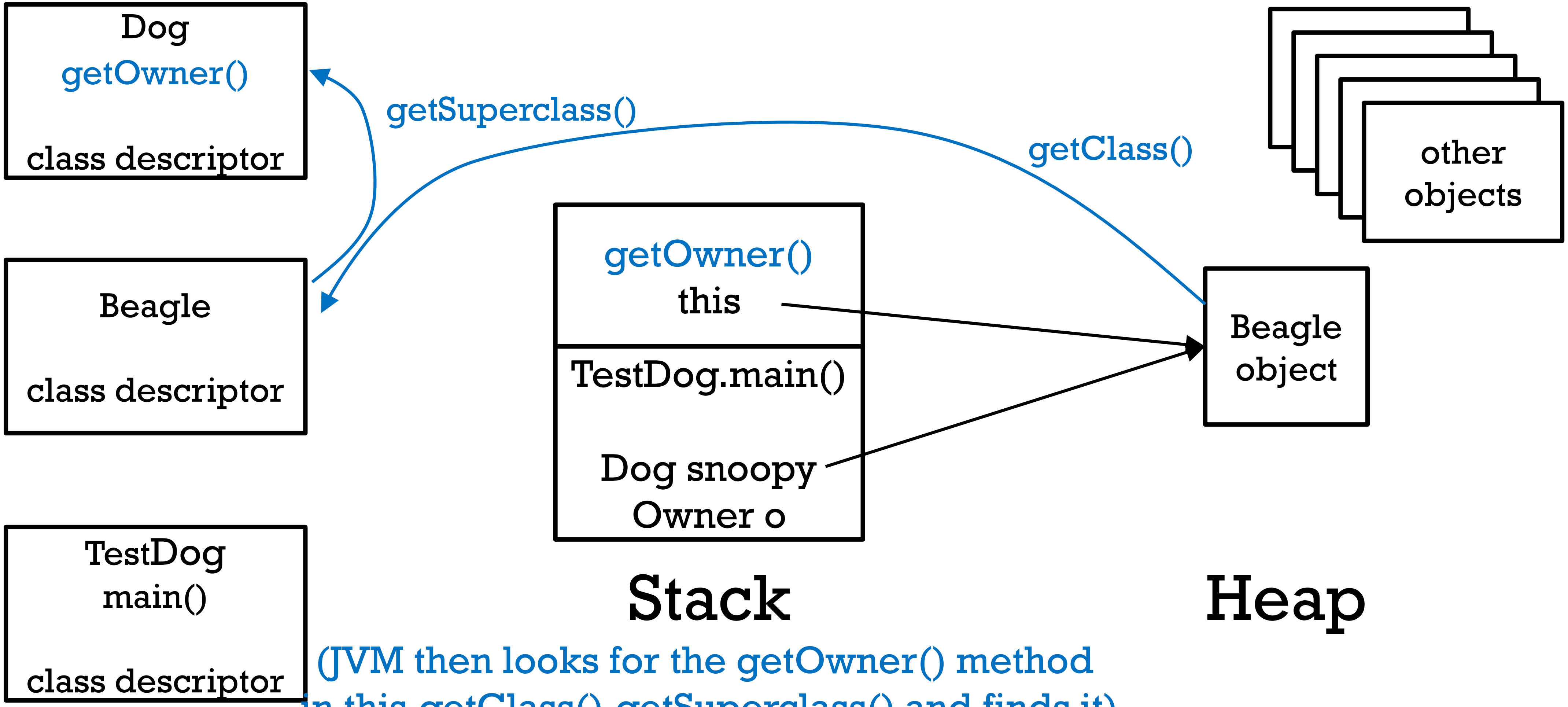
Beagle object

Stack

Heap

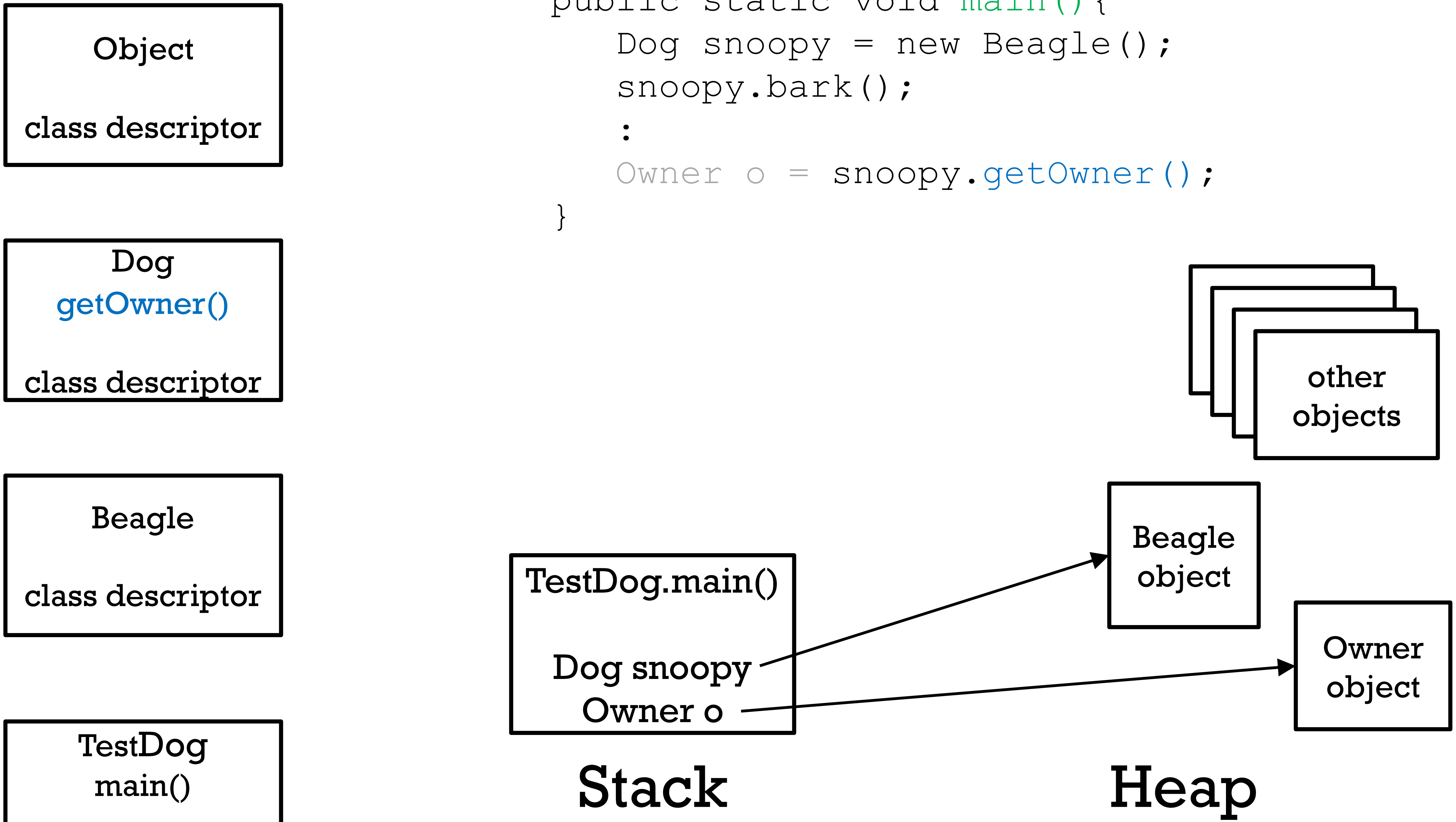
(JVM then looks for the getOwner() method in this.getClass() and doesn't finds it)

```
public static void main(){
    Dog snoopy = new Beagle();
    snoopy.bark();
    :
    Owner o = snoopy.getOwner();
}
```

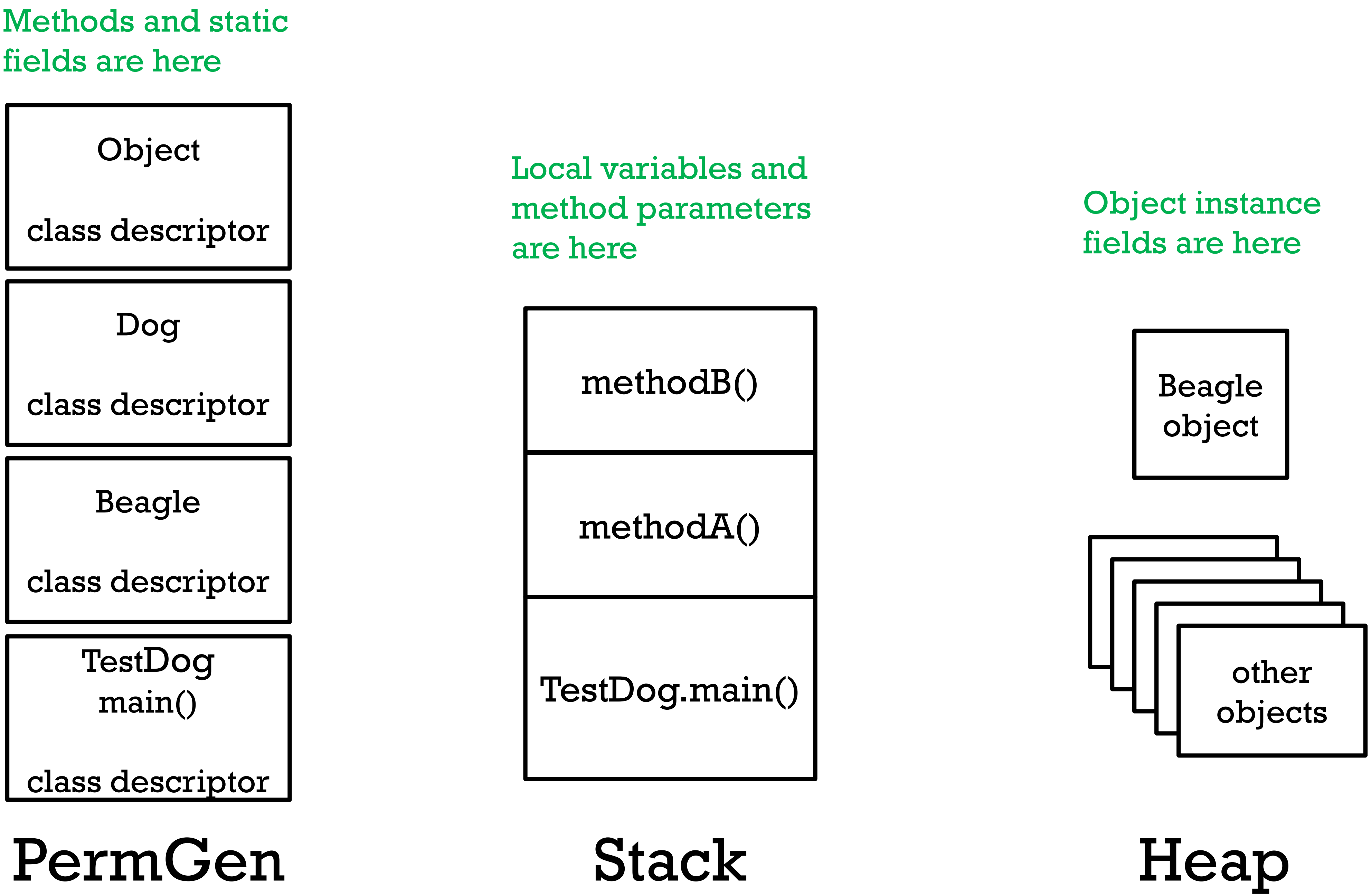


(JVM then looks for the getOwner() method in this.getClass().getSuperclass() and finds it)

```
public static void main(){
    Dog snoopy = new Beagle();
    snoopy.bark();
    :
    Owner o = snoopy.getOwner();
}
```



(getOwner terminates,
reference assigned to local variable)



Methods and static fields are here

Object

class descriptor

Dog

class descriptor

Beagle

class descriptor

TestDog main()

class descriptor

PermGen

Local variables and method parameters are here

methodB()

methodA()

TestDog.main()

Stack

Object instance fields are here

Beagle object

other objects

Heap

- Recursion