UCC
Coláiste na hOllscoile Corcaigh
University College Cork, Ireland

Java Classes and Constructors

Dr. Krishnendu Guha

Assistant Professor/ Lecturer

School of Computer Science and Information Technology

University College Cork

Email: kguha@ucc.ie

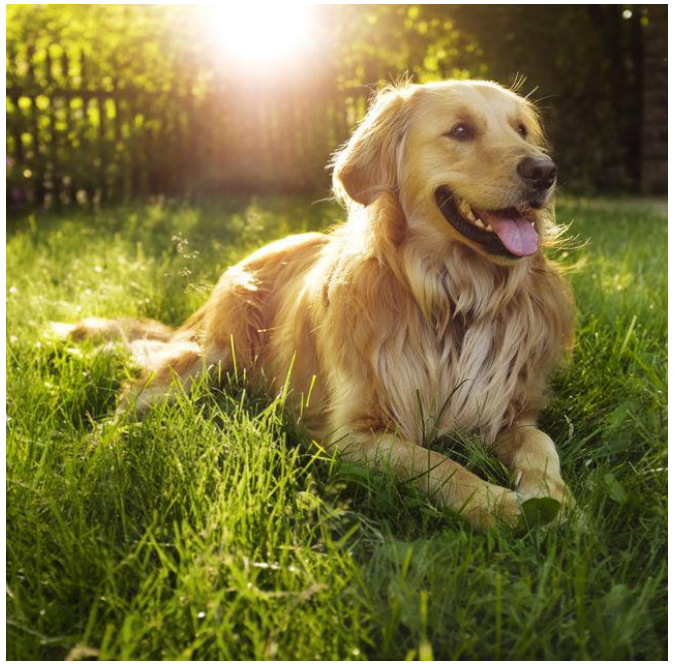
Recap

- We've been creating objects using class definitions from the Java SE class library, which consists of nearly 4000 classes in Java 12!
- However, **we often need to create custom classes**.
- It is **possible to place more than one class in a file**.
- However, **only one class (top-level) can be public**. We will use: one class per file.

Previous Example

```
public class Student {
	String name;
	String hairColour;
	double height;
	String degree;
	void introduce() {
		System.out.println("Dear stranger, I am " + name + ". Nice to meet you.");
	}
	void changeHairColour(String newHairColour) {
		hairColour = newHairColour;
		System.out.println("I dyed my hair to " + newHairColour);
	}
}
```

Another Example



- What are the attributes of a dog?
- What are the behaviors a dog may have?

The minimal class definition:

```
class Dog {
}
```

Then, we **add some attributes** and **behaviours to a class**.

A **class** can also be ***called a logical template to create the objects that share common properties and methods***.

dog class: Dog.java

```
public class Dog {
    // instance variables
    public String name;
    public int age;
    public String breed;
    ...
    // constructor to instantiate
    public Dog() {
    }
    // methods
    public void bark() {
    // bark
    }
    public void setName(String new_name) {
    // change the name
    }
    ...
}
```

Test Class: DogTester.java

```
class DogTester {
public static void main(String[] args) {
        Dog d1 = new Dog();
        Dog d2 = new Dog();
        d1.bark();
        d2.bark();
        d1.setName("Charlie");
        d2.setName("Teddy");
        d1.setAge(2);
        d2.setAge(7);
        d1.setBreed("Kangal Shepherd");
        d2.setBreed("Pug");

        int age_of_d1 = d1.getAge();
        System.out.println(age_of_d1);
        System.out.println(d1.getAge());
        System.out.println(d2.getName() + " is a " + d2.getAge() +
" year old " + d2.getBreed());
        System.out.println(d1.toString());
        System.out.println(d1);
        }
}
```

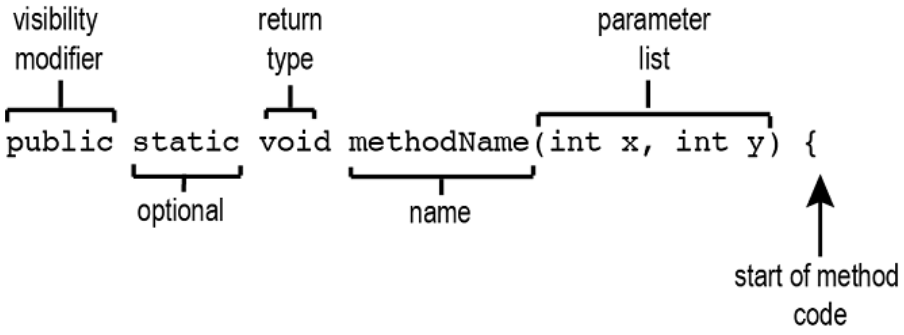
Interface of Dog

- public void bark()
- public void setName(String s)
- public void setAge(int n)
- public void setBreed(String s)
- public String getName()
- public int getAge()
- public String getBreed()
- public String toString()

Method Signatures

The signature of a method, tells us:

- method name
- how many parameters, in what order and what type of data each must be
- the type of the result that it returns, if anything



Some special methods

Getters and Setters: Informally, we refer to some instance methods as getters and others as setters.

- Q: Which of Dog's instance methods are getters?
- Q: Which are setters?
- Q: Which one is neither?
- Q: What can we say about the signatures of getters versus the signatures of setters?

toString(): This method returns the String representation of the object.

Variables

Local variables

- Declared within the **body of a method**
- Used to **store something temporarily** during the body of the method

Instance variables

- Declared at **'top-level'** within a class definition, not within the body of a method
- Used to **store the 'state' of an object**
- **Each object has its own copy** of the instance variables

Variable Scope

Local variables

- block scope in Java

Instance variables

- everywhere!
- but outside the class definition you must prefix with an object reference
- Example: the following is an error – think about why

```
public class DogTester {
	public static void main(String[] args) {
		Dog d1 = new Dog();
		Dog d2 = new Dog();
		name = "Charles";
	}
}
```

```java
public class DogTester {
    public static void main(String[] args) {
        Dog d1 = new Dog();
        Dog d2 = new Dog();
        d1.name = "Charles";
    }
}
```

Variable Initialization

Local variables

- You must **explicitly initialize them yourself**, else **compile-time error** in Java

Instance variables

- **Java initializes them** with **default values**
- But **you can initialize them yourself** in the declaration
- Or **you can define one or more constructors**

Default initial values

Data type	**Default value**
int	0
double	0.0
boolean	false
String, Scanner, Random, Dog, ...	null

Constructors: why?

So far, **we have to create an object** and **then use setters** to **set its state to the values that we want**, e.g.:

```
public static void main(String[] args) {
    Dog d1 = new Dog();
    d1.setName("Charles");
    d1.setAge(2);
    d1.setBreed("Dalmatian");
    // etc.
}
```

More convenient would be to supply the values when we create the object, e.g.:

```
public static void main(String[] args) {
    Dog d1 = new Dog("Charles", 2, "Dalmatian");
    // etc.
}
```


Any Questions


Thank you!