JAVA PROGRAMMEREN – LES 5:

ACCESS MODIFIERS
OVERLOADING
STATIC
INTERFACES

Robert-Jan Elias

robert-jan.elias@novi-education.nl

NOVI

AGENDA

- Terugblik
- Access modifiers
- Overloading
- Static keyword
- Interfaces
- Workshop
- Oefenopdracht
- Huiswerk

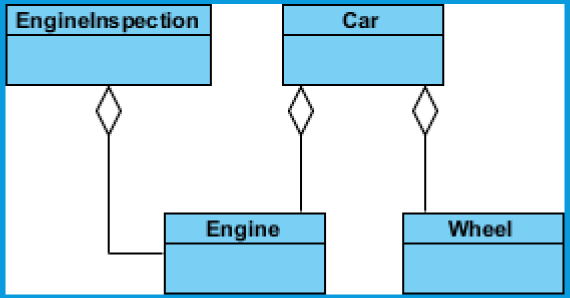
TERUGBLIK

- Aggregatie & compositie
- Overerving
- Abstracte klassen
- Polymorfisme

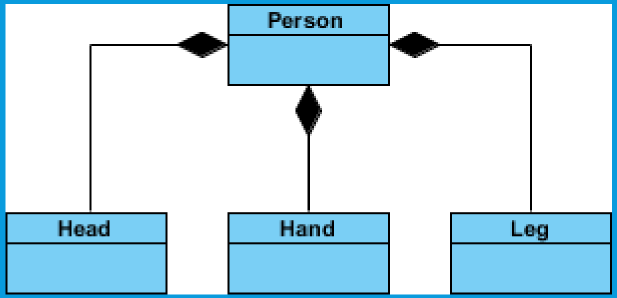
NOVI

SOORTEN RELATIES (UML NOTATIE)

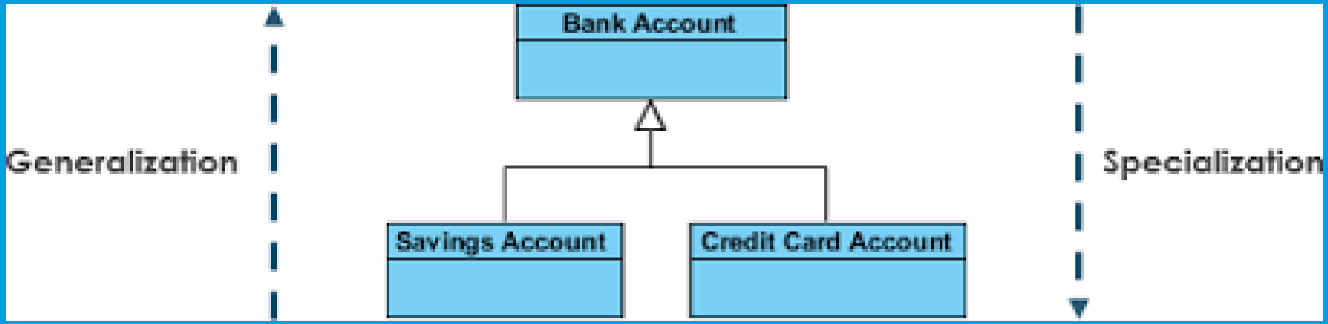
Aggregatie:



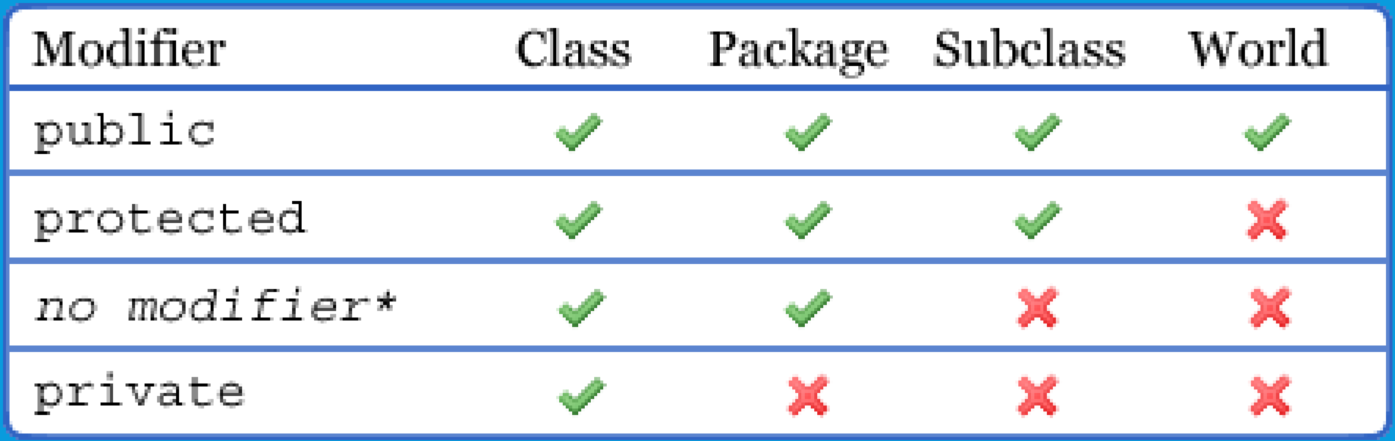
Compositie:



Overerving:



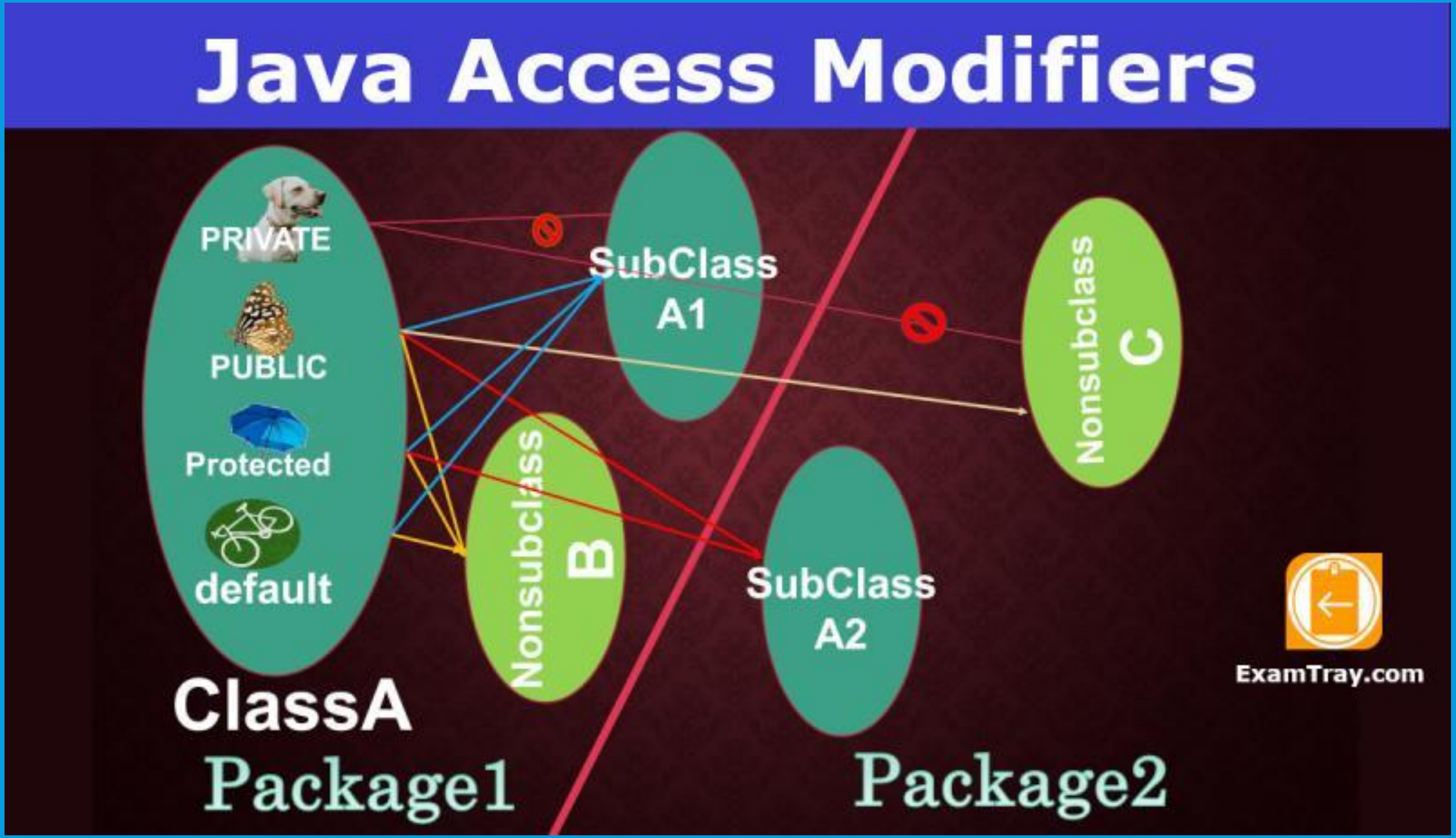
ACCESS MODIFIERS



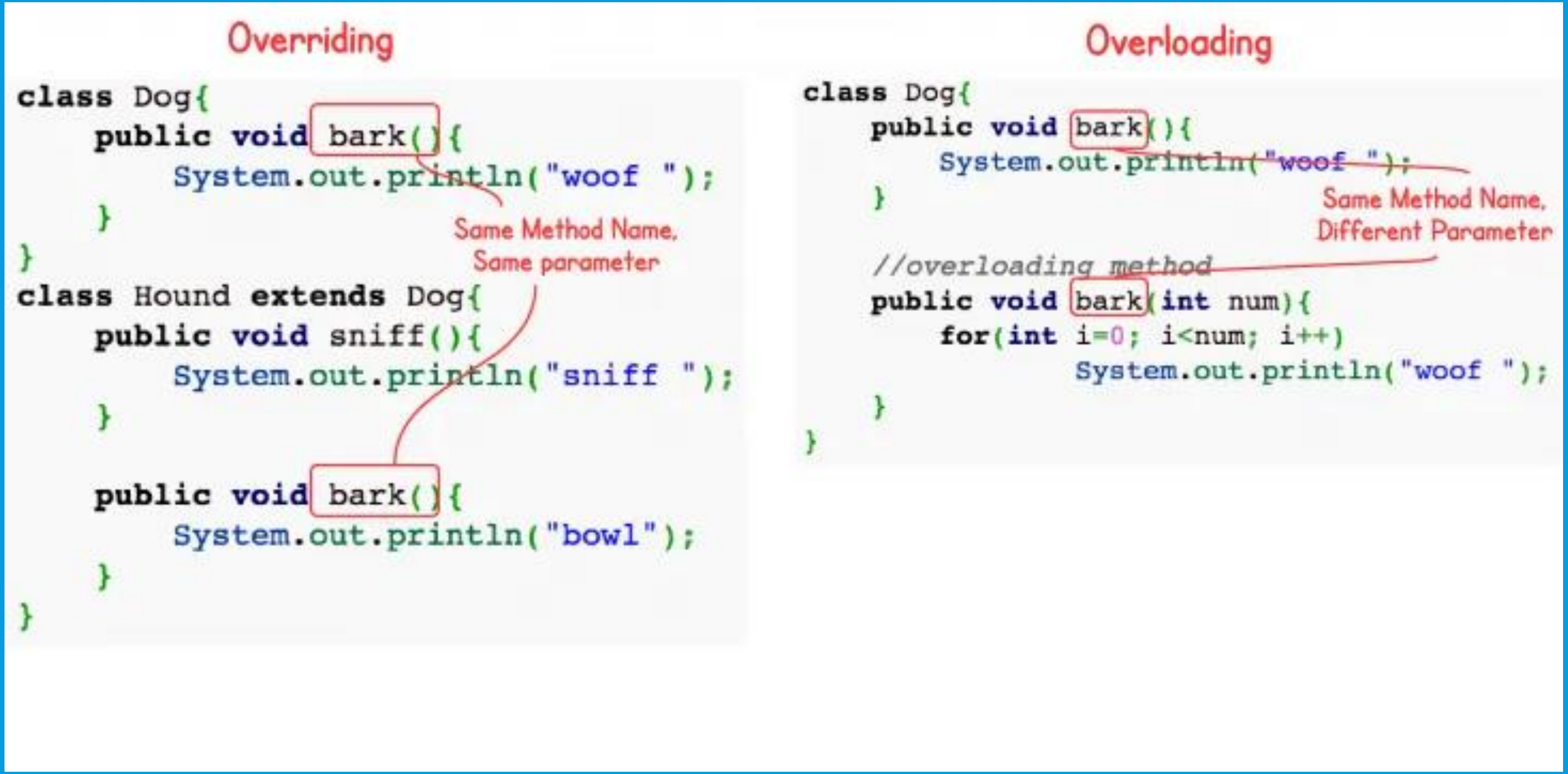
Modifier	Class	Package	Subclass	World
`public`	✓	✓	✓	✓
`protected`	✓	✓	✓	✗
*no modifier**	✓	✓	✗	✗
`private`	✓	✗	✗	✗

NOVI

ACCESS MODIFIERS



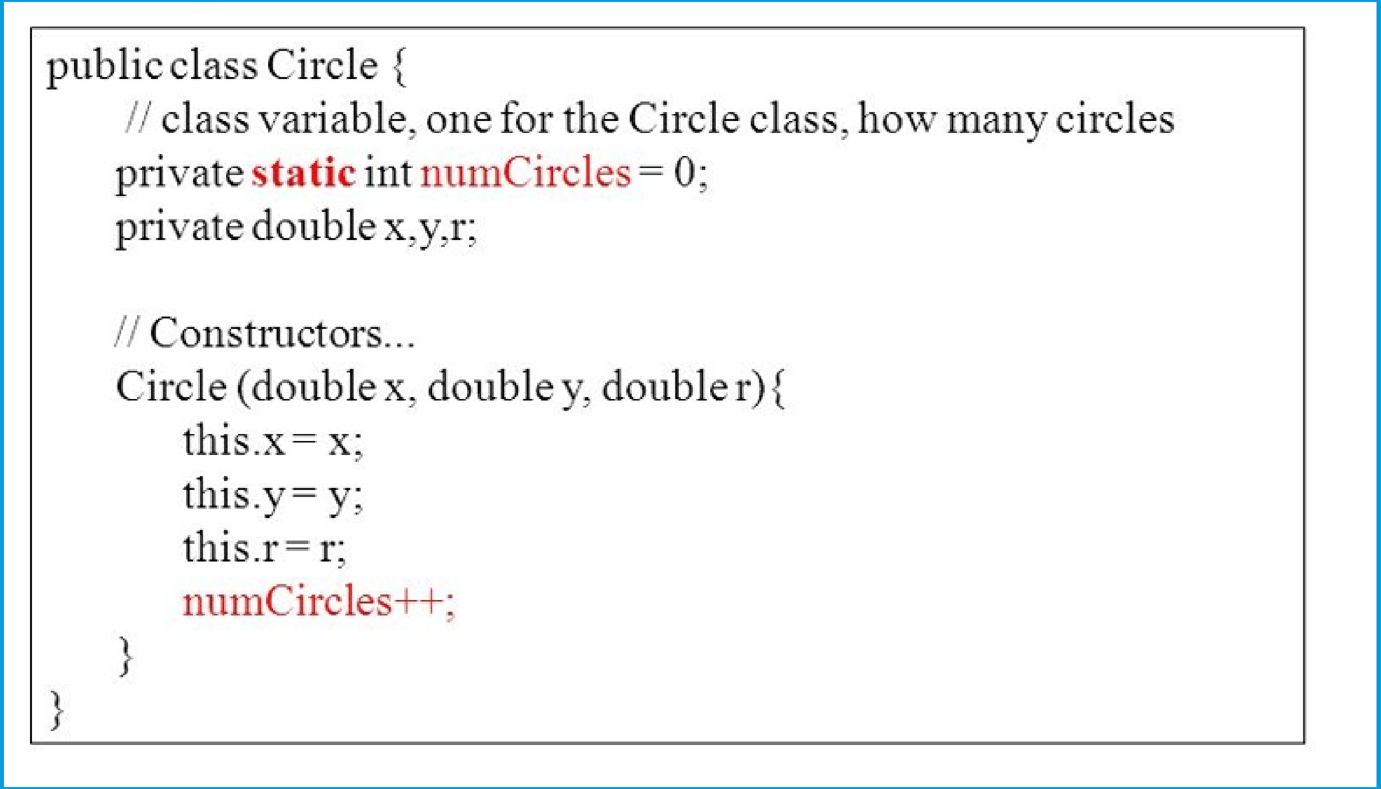
OVERLOADING



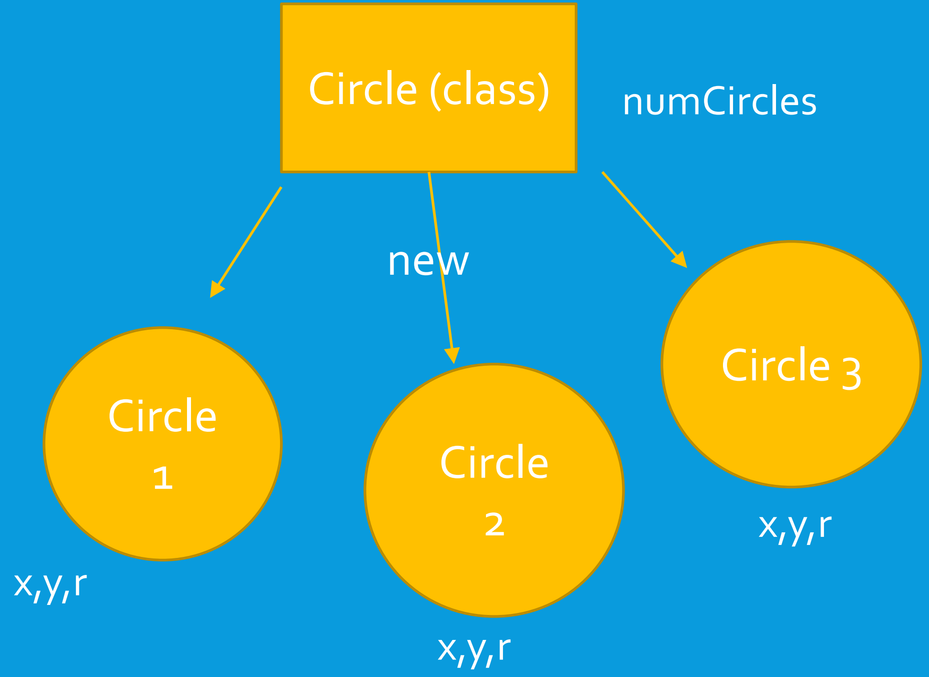
```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }
}
class Hound extends Dog{
    public void sniff(){
        System.out.println("sniff ");
    }

    public void bark(){
        System.out.println("bowl");
    }
}
```

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }

    //overloading method
    public void bark(int num){
        for(int i=0; i<num; i++)
            System.out.println("woof ");
    }
}
```

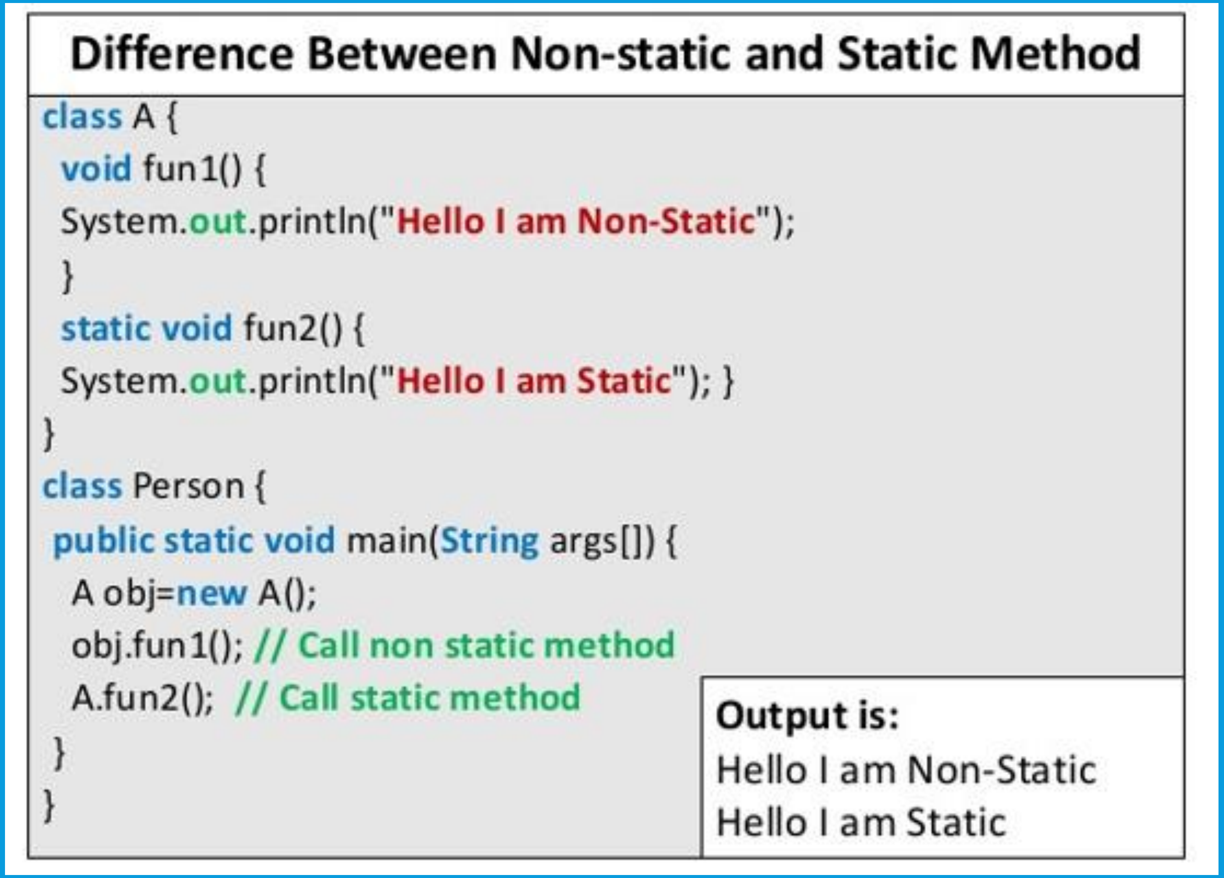
STATIC VARIABLES



```
public class Circle {
     // class variable, one for the Circle class, how many circles
    private static int numCircles = 0;
    private double x,y,r;

    // Constructors...
    Circle (double x, double y, double r){
        this.x = x;
        this.y = y;
        this.r = r;
        numCircles++;
    }
}
```



STATIC METHODS

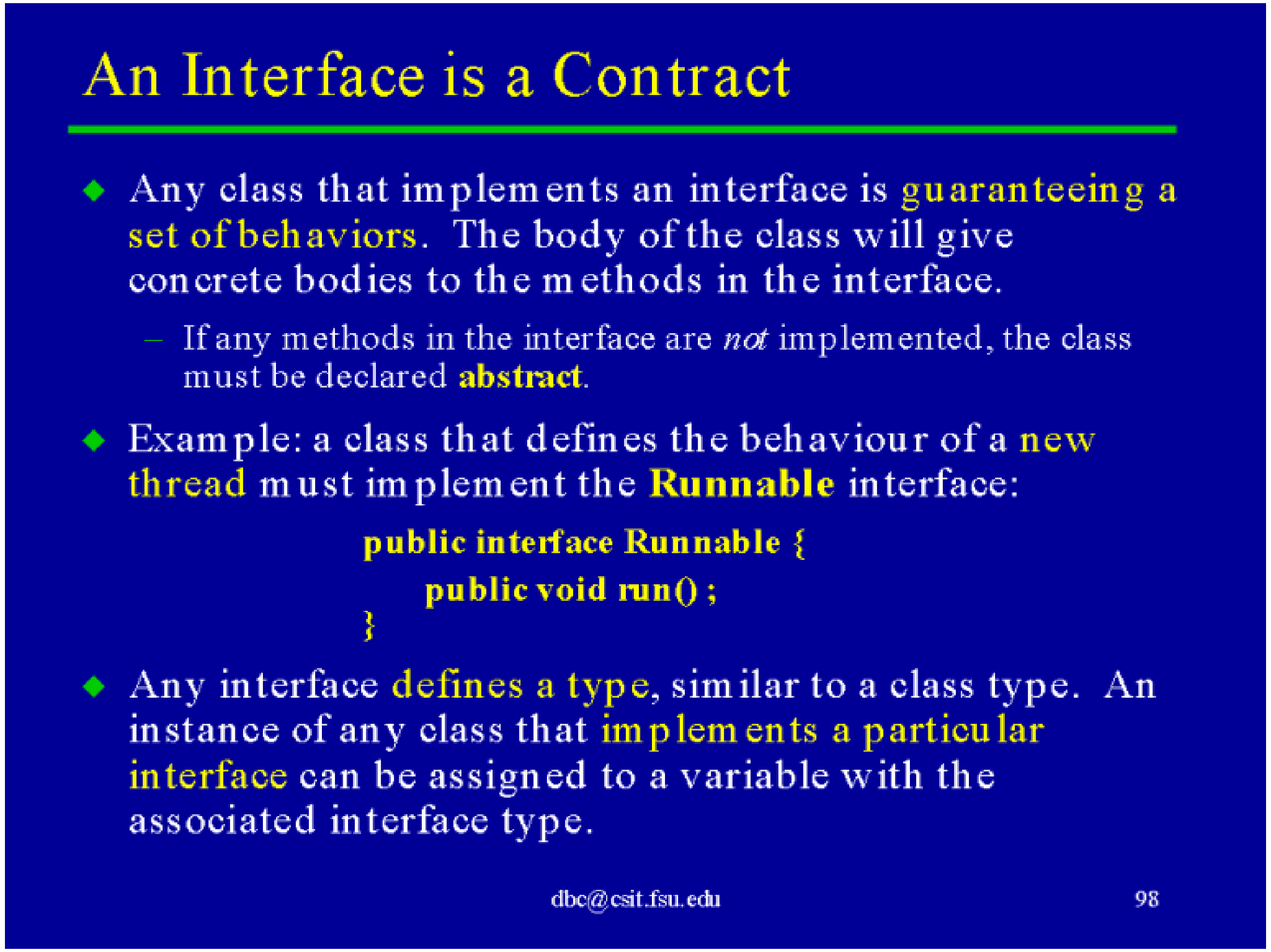


Difference Between Non-static and Static Method

```
class A {
 void fun1() {
 System.out.println("Hello I am Non-Static");
 }
 static void fun2() {
 System.out.println("Hello I am Static"); }
}
class Person {
 public static void main(String args[]) {
  A obj=new A();
  obj.fun1(); // Call non static method
  A.fun2();  // Call static method
 }
}
```

Output is:

```
Hello I am Non-Static
Hello I am Static
```



An Interface is a Contract

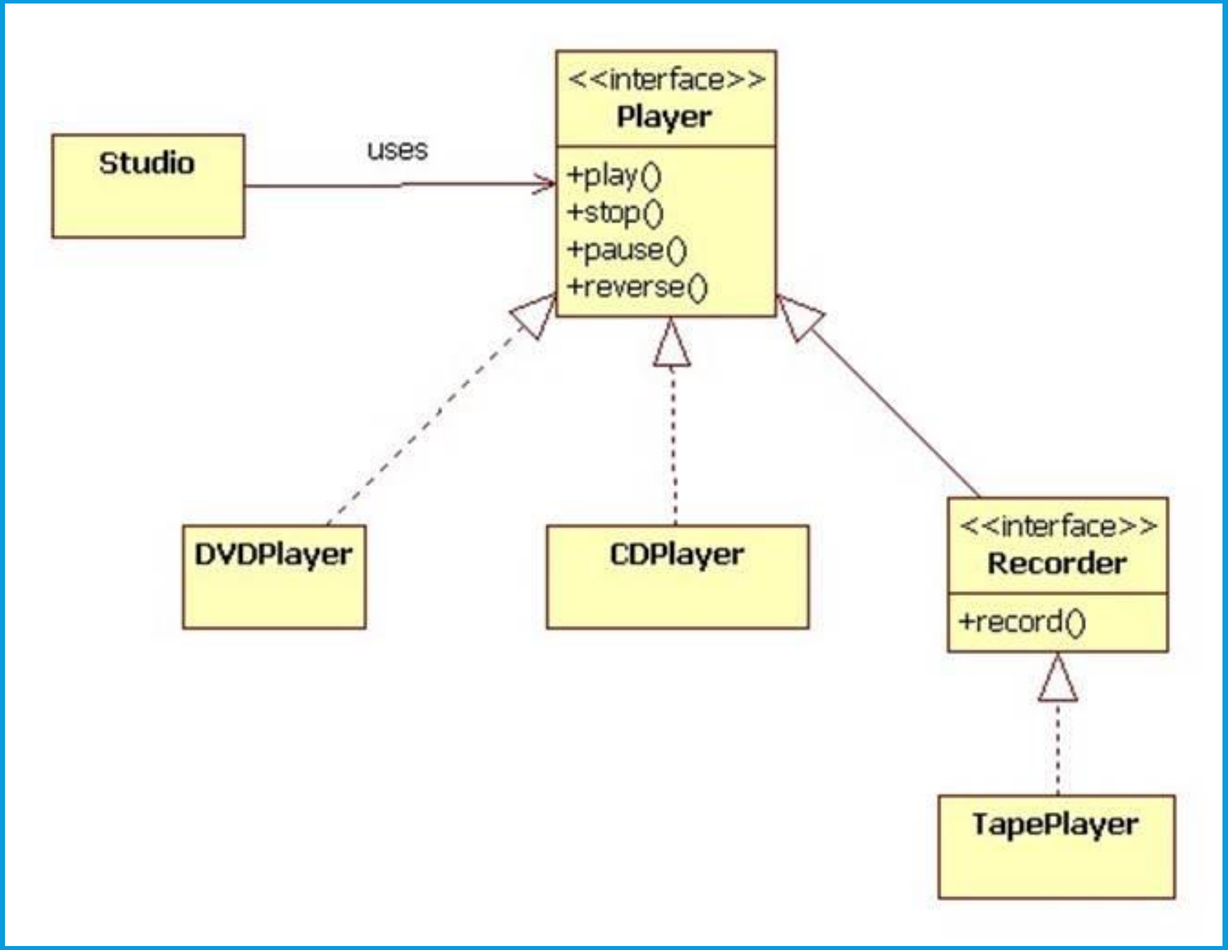
- Any class that implements an interface is guaranteeing a set of behaviors. The body of the class will give concrete bodies to the methods in the interface.
  - If any methods in the interface are *not* implemented, the class must be declared **abstract**.
- Example: a class that defines the behaviour of a new thread must implement the **Runnable** interface:

```
public interface Runnable {
    public void run() ;
}
```

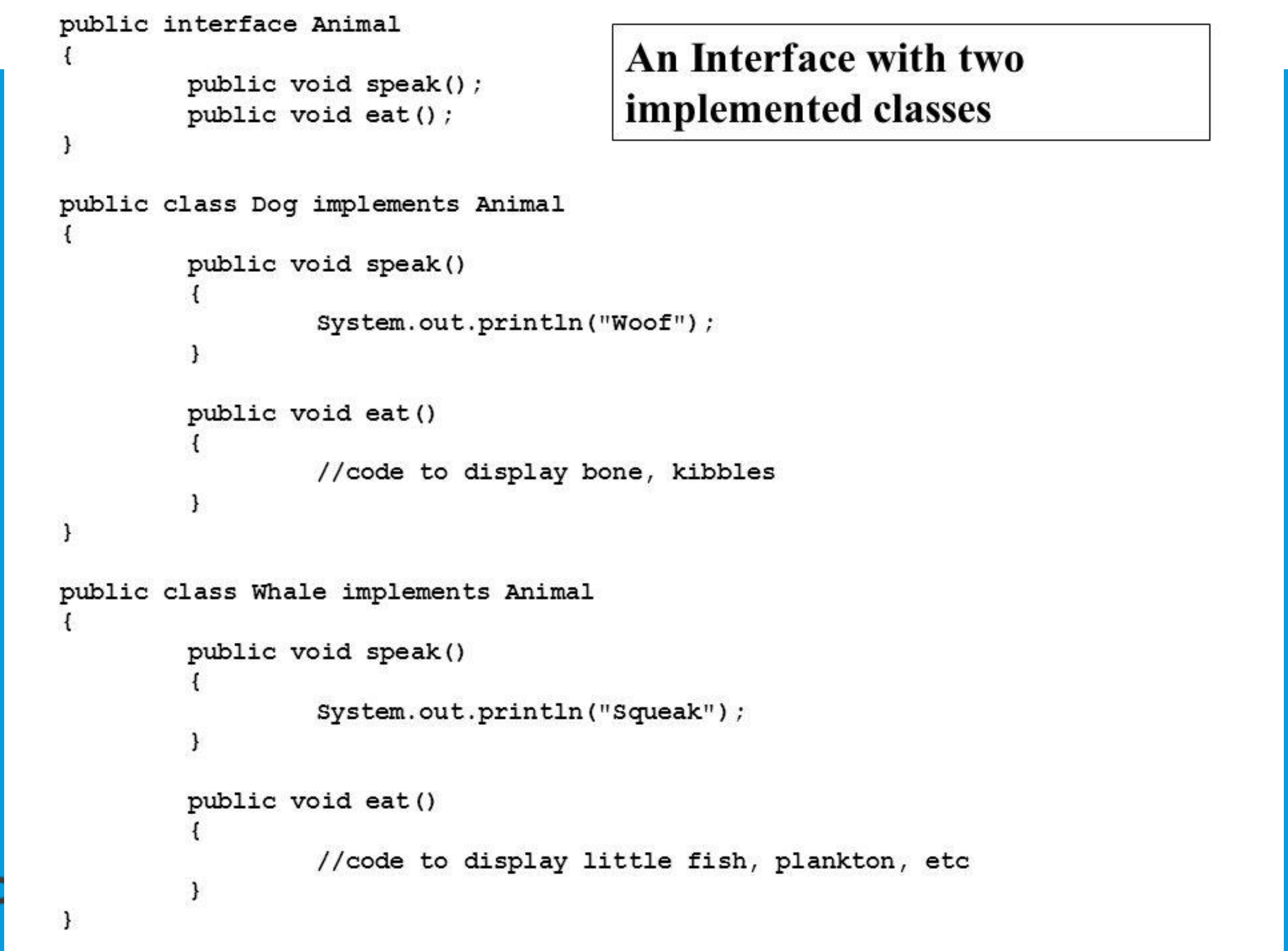
- Any interface defines a type, similar to a class type. An instance of any class that implements a particular interface can be assigned to a variable with the associated interface type.

INTERFACES

INTERFACES (UML NOTATIE)



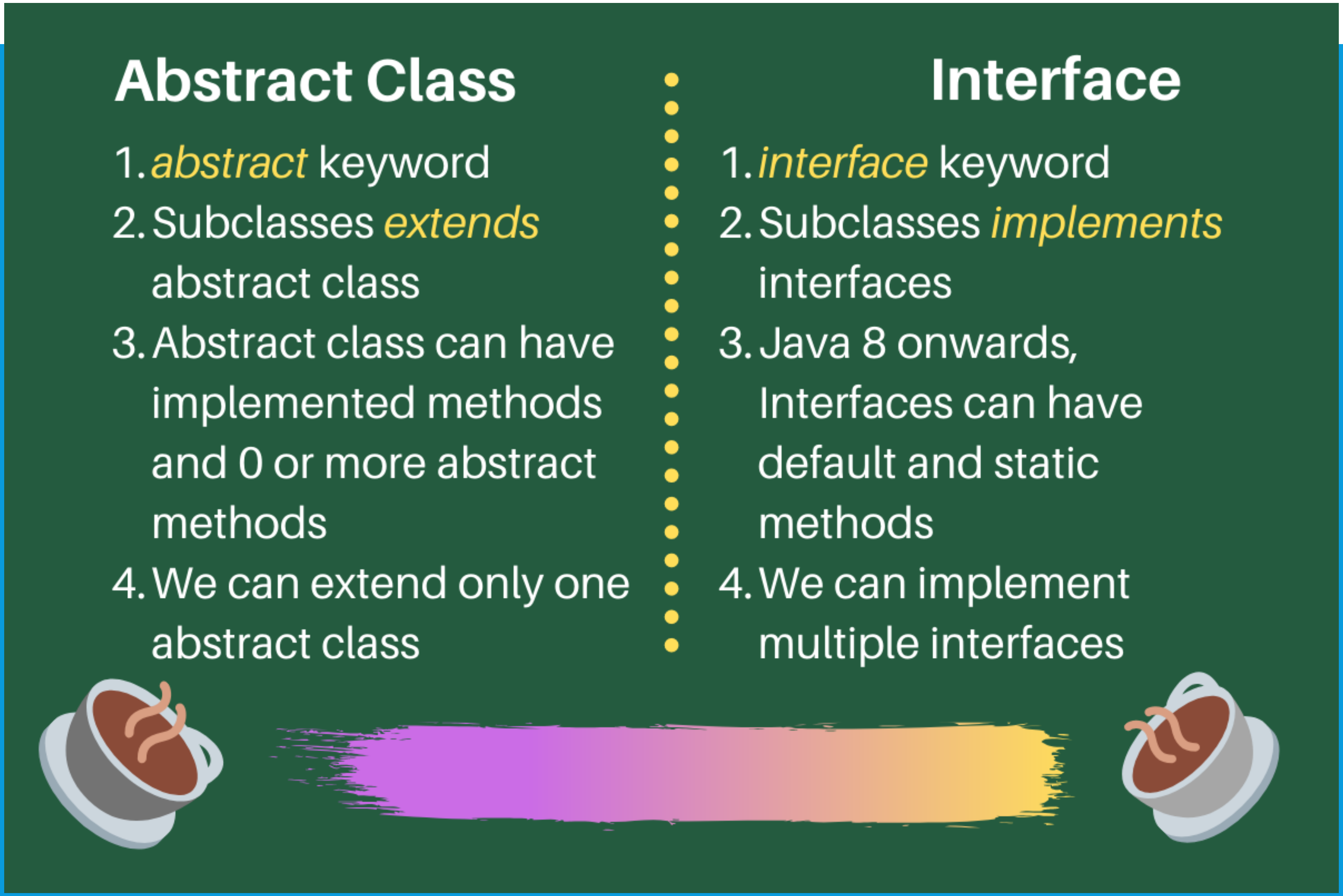
INTERFACES



An Interface with two implemented classes

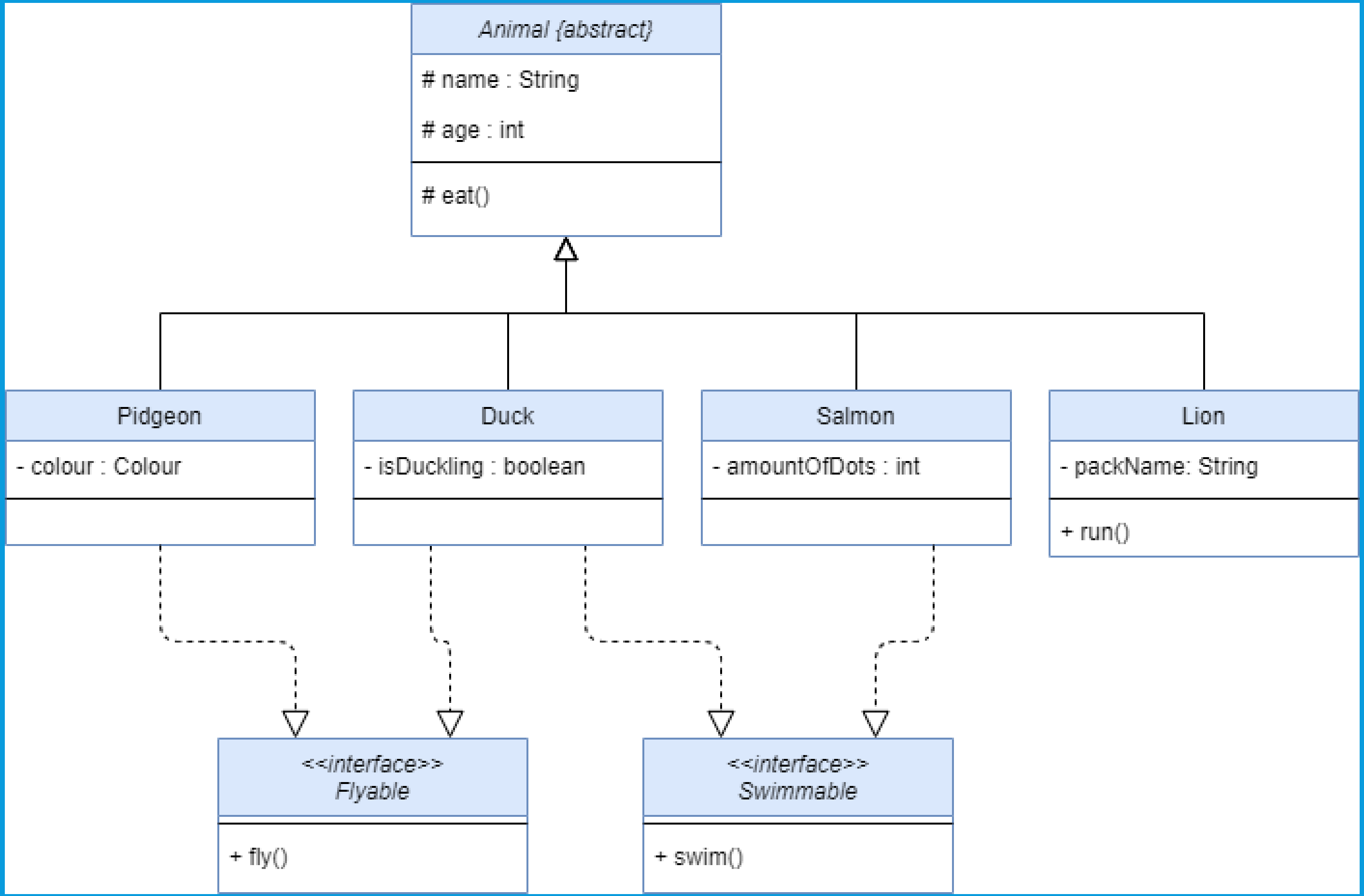
```
public interface Animal
{
        public void speak();
        public void eat();
}

public class Dog implements Animal
{
        public void speak()
        {
                System.out.println("Woof");
        }

        public void eat()
        {
                //code to display bone, kibbles
        }
}

public class Whale implements Animal
{
        public void speak()
        {
                System.out.println("Squeak");
        }

        public void eat()
        {
                //code to display little fish, plankton, etc
        }
}
```

ABSTRACT CLASSES VERSUS INTERFACES



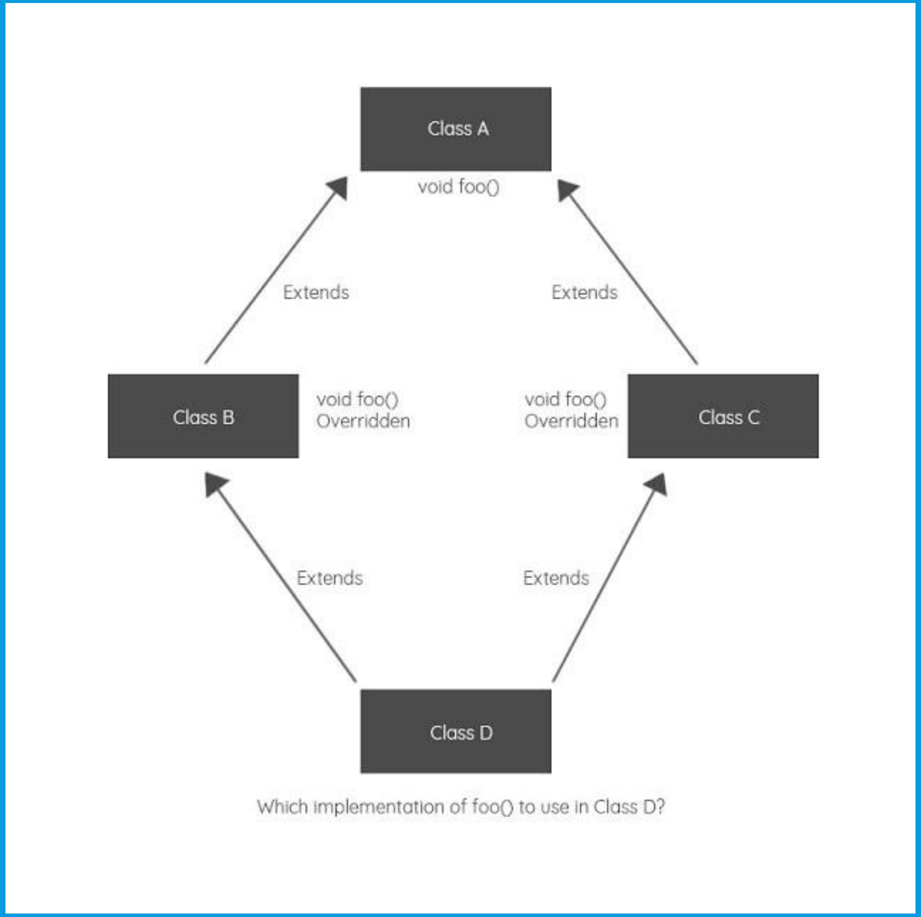
Abstract Class	Interface
1. *abstract* keyword	1. *interface* keyword
2. Subclasses *extends* abstract class	2. Subclasses *implements* interfaces
3. Abstract class can have implemented methods and 0 or more abstract methods	3. Java 8 onwards, Interfaces can have default and static methods
4. We can extend only one abstract class	4. We can implement multiple interfaces

ABSTRACT CLASSES & INTERFACES

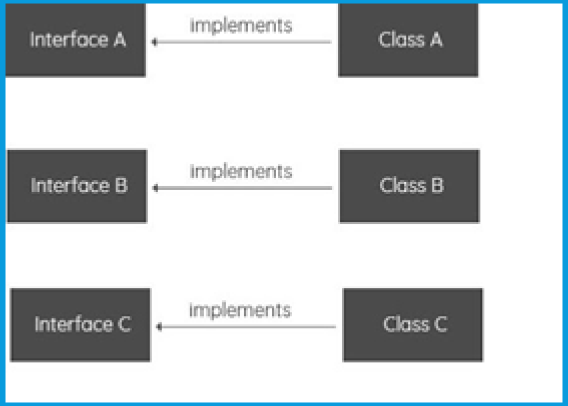


NOVI

MULTIPLE INHERITANCE & INTERFACES



Which implementation of foo() to use in Class D?



```
interface A
{
 void foo();
}

interface B
{
 void foo();
}

interface C
{
 void foo();
}

Class D implements B,C
{
 void foo()
     {
         Print("Hello Everybody");
     }
}
```

WORKSHOP

- Access modifiers
- Method overloading
- Interfaces

Fly & Drive

- Maak een nieuw Console project aan in IntelliJ.
- Voeg een *Main* class toe met een main() method.

- Maak een abstracte class *Vehicle* aan met de volgende velden en (abstract) methods:
  - Speed (integer)
  - Weight (float)
  - startEngine()
  - turnOffEngine()

- Maak de volgende interfaces:
  - *Flyable* met methods takeOff(), land(), changeHeight().
  - *Driveable* met methods: accelerate(), brake(), changeGear().

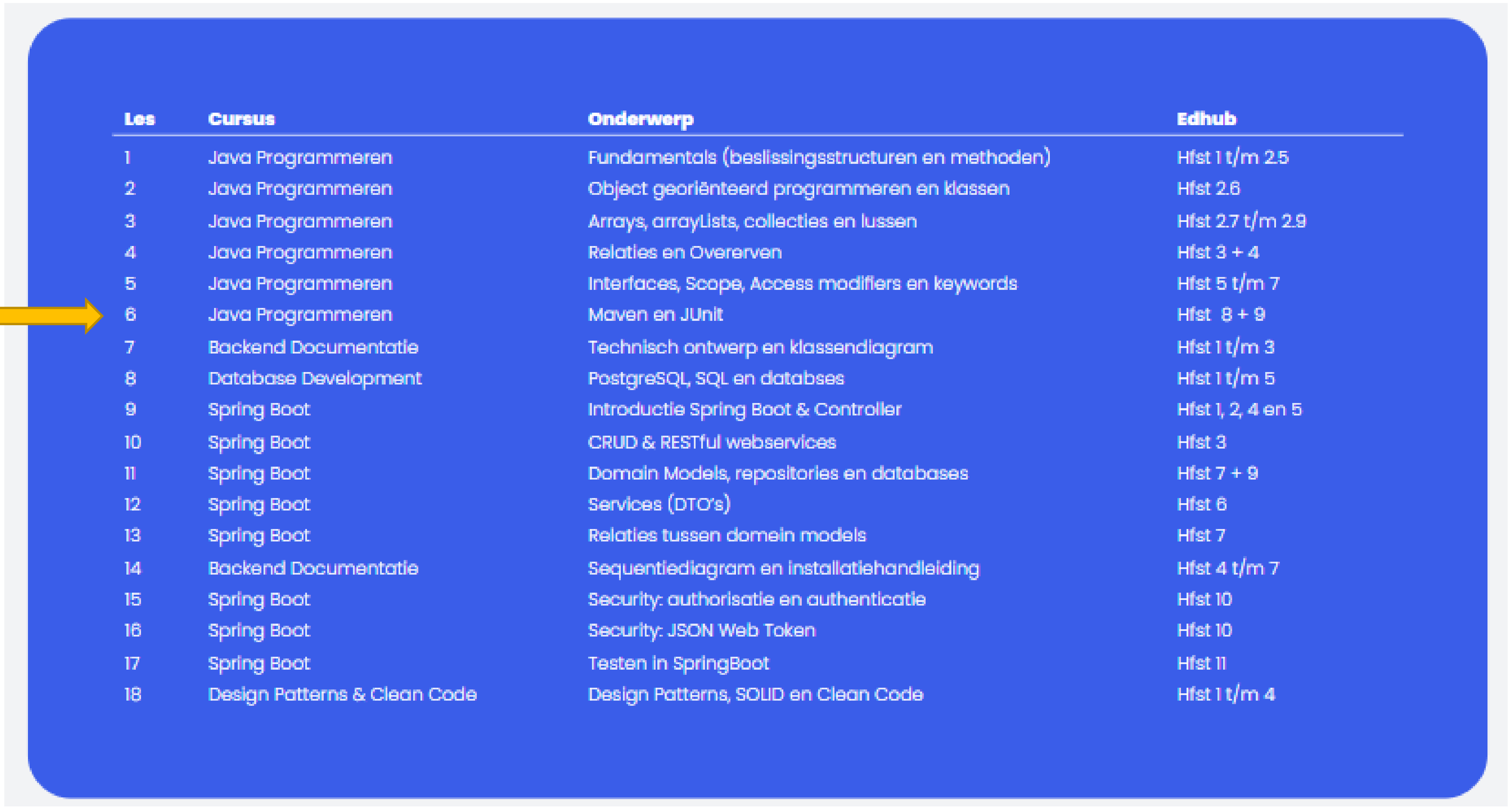
- Maak de volgende afgeleide subclasses en gebruik de juiste interfaces:
  - *Car*
  - *Plane*
  - *FlyingCar*

- Instantieer objecten voor deze subclasses vanuit main() en laat ze rijden en vliegen.

OEFENOPDRACHT

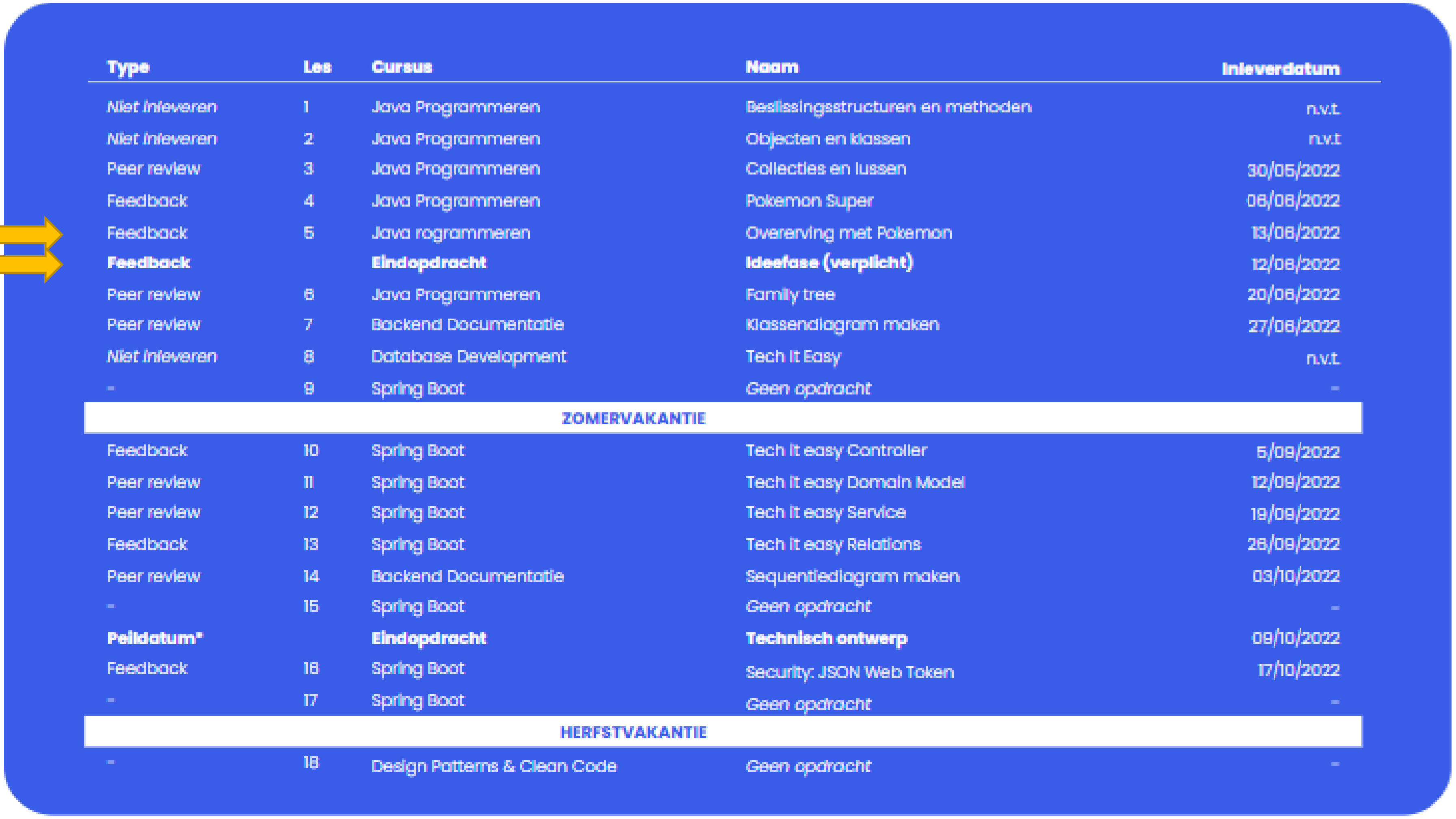
NOVI

HUISWERK - LEZEN



Les	Cursus	Onderwerp	Edhub
1	Java Programmeren	Fundamentals (beslissingsstructuren en methoden)	Hfst 1 t/m 2.5
2	Java Programmeren	Object georiënteerd programmeren en klassen	Hfst 2.6
3	Java Programmeren	Arrays, arrayLists, collecties en lussen	Hfst 2.7 t/m 2.9
4	Java Programmeren	Relaties en Overerven	Hfst 3 + 4
5	Java Programmeren	Interfaces, Scope, Access modifiers en keywords	Hfst 5 t/m 7
6	Java Programmeren	Maven en JUnit	Hfst 8 + 9
7	Backend Documentatie	Technisch ontwerp en klassendiagram	Hfst 1 t/m 3
8	Database Development	PostgreSQL, SQL en databses	Hfst 1 t/m 5
9	Spring Boot	Introductie Spring Boot & Controller	Hfst 1, 2, 4 en 5
10	Spring Boot	CRUD & RESTful webservices	Hfst 3
11	Spring Boot	Domain Models, repositories en databases	Hfst 7 + 9
12	Spring Boot	Services (DTO's)	Hfst 6
13	Spring Boot	Relaties tussen domein models	Hfst 7
14	Backend Documentatie	Sequentiediagram en installatiehandleiding	Hfst 4 t/m 7
15	Spring Boot	Security: authorisatie en authenticatie	Hfst 10
16	Spring Boot	Security: JSON Web Token	Hfst 10
17	Spring Boot	Testen in SpringBoot	Hfst 11
18	Design Patterns & Clean Code	Design Patterns, SOLID en Clean Code	Hfst 1 t/m 4

HUISWERK - MAKEN



Type	Les	Cursus	Naam	Inleverdatum
Niet inleveren	1	Java Programmeren	Beslissingsstructuren en methoden	n.v.t.
Niet inleveren	2	Java Programmeren	Objecten en klassen	n.v.t.
Peer review	3	Java Programmeren	Collecties en lussen	30/05/2022
Feedback	4	Java Programmeren	Pokemon Super	06/06/2022
Feedback	5	Java rogrammeren	Overerving met Pokemon	13/06/2022
Feedback		**Eindopdracht**	**Ideefase (verplicht)**	12/06/2022
Peer review	6	Java Programmeren	Family tree	20/06/2022
Peer review	7	Backend Documentatie	Klassendiagram maken	27/06/2022
Niet inleveren	8	Database Development	Tech It Easy	n.v.t.
-	9	Spring Boot	*Geen opdracht*	-
		ZOMERVAKANTIE		
Feedback	10	Spring Boot	Tech It easy Controller	5/09/2022
Peer review	11	Spring Boot	Tech It easy Domain Model	12/09/2022
Peer review	12	Spring Boot	Tech It easy Service	19/09/2022
Feedback	13	Spring Boot	Tech It easy Relations	26/09/2022
Peer review	14	Backend Documentatie	Sequentiediagram maken	03/10/2022
-	15	Spring Boot	*Geen opdracht*	-
Peildatum*		**Eindopdracht**	**Technisch ontwerp**	09/10/2022
Feedback	16	Spring Boot	Security: JSON Web Token	17/10/2022
-	17	Spring Boot	*Geen opdracht*	-
		HERFSTVAKANTIE		
-	18	Design Patterns & Clean Code	*Geen opdracht*	-