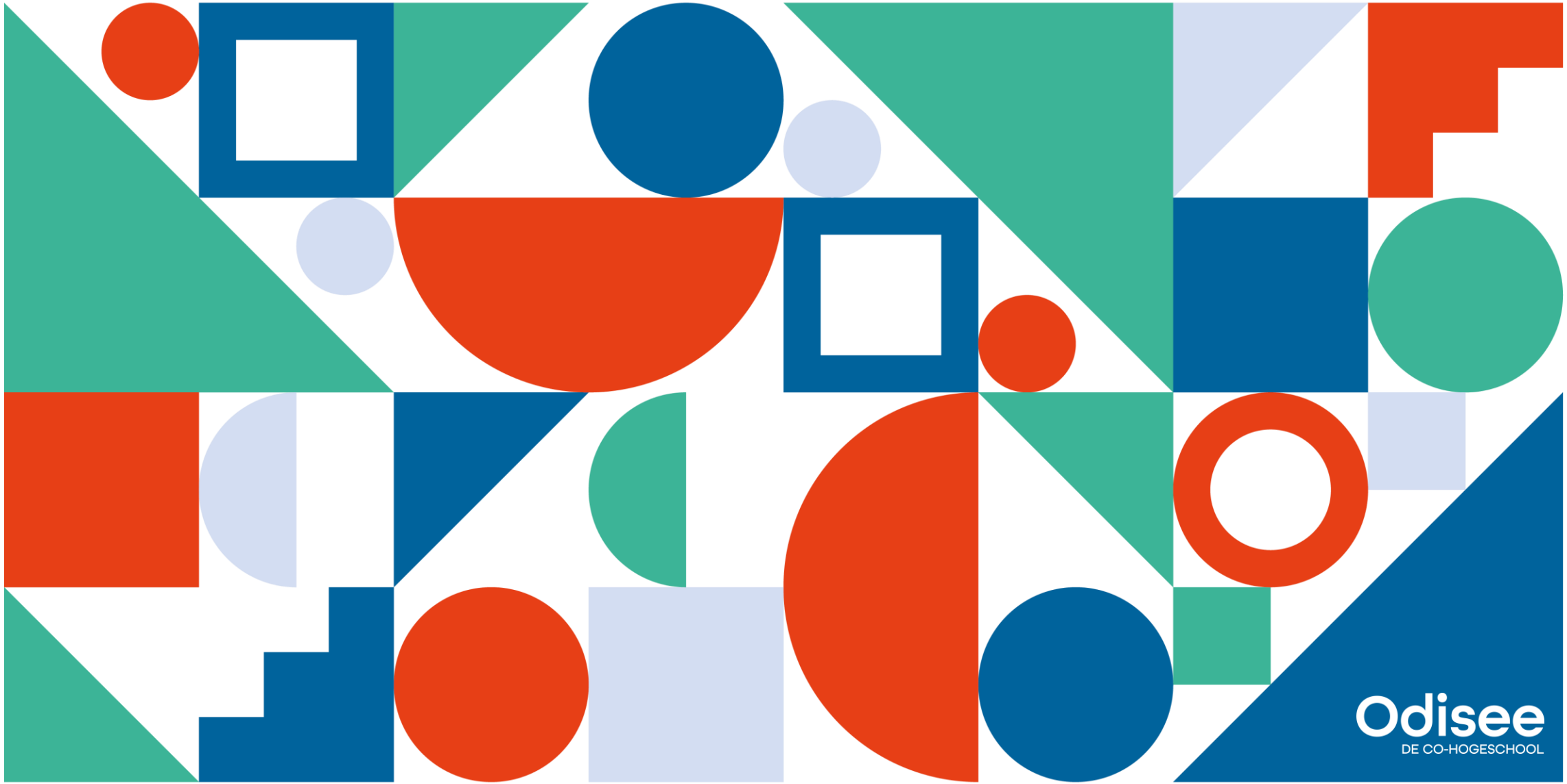
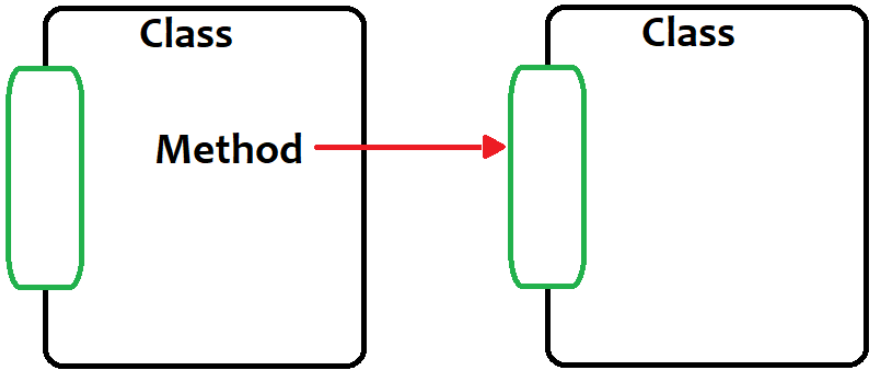

Odisee
DE CO-HOGESCHOOL

Interfaces & Exceptions

Odisee
DE CO-HOGESCHOOL

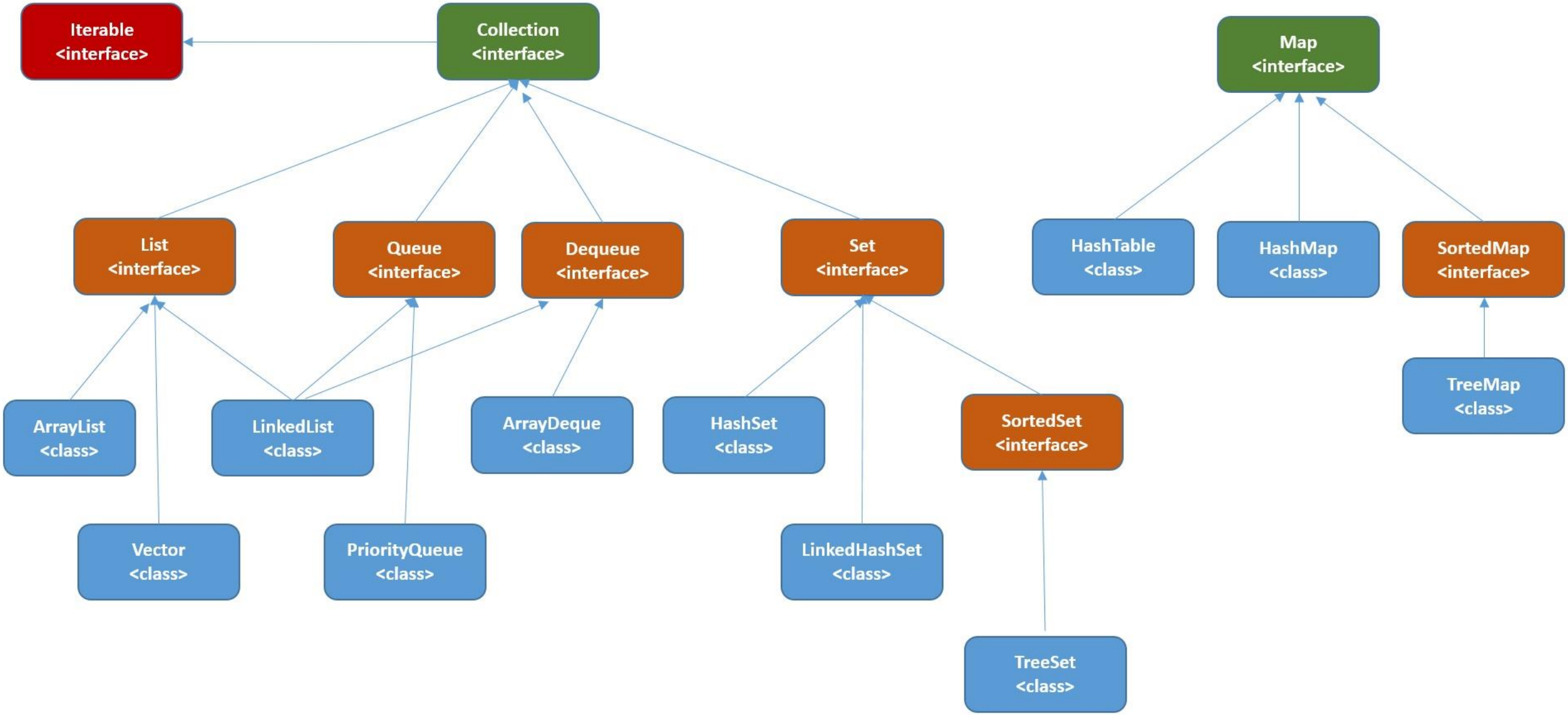
Interfaces

- Een interface is een koppelvlak waarmee twee systemen met elkaar communiceren
  - GUI: Grafische User Interface
    - Communicatie tussen user en interface
  - Binnen een programmeertaal
    - Communicatie tussen twee klassen/objecten
    - De interface van een klasse = declaratie/definitie van alle publieke methoden van de klasse



- Interfaces bestaan in vele gedaantes in informatica
  - UI
  - REST API
  - Meerdere java libraries (zie Oracle vs Google)
  - Linux kernel header files
- In java, C# zijn interfaces een keyword met dezelfde invulling:
  - = de declarative van de publieke methods
  - Classes implementeren interfaces door de code voor de methods te leveren

Collection Framework Hierarchy

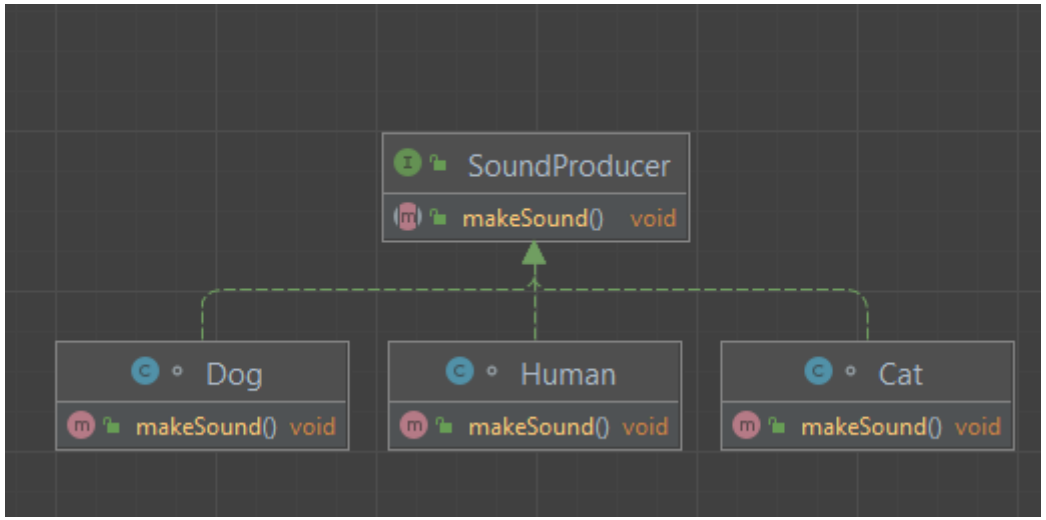


Voorbeeld Interface

```
public interface SoundProducer {
    void makeSound();
}

class Dog implements SoundProducer{
    @Override public void makeSound() {
        System.out.println("Woof");
    }
    public void fetch() { … }
}

class Human implements SoundProducer{
    @Override public void makeSound() {
        System.out.println("Hallo");
    }
}
```

- Gelijkaardig aan abstacte class
- Keyword implements iso extends
- Geen (private, protected, …) visibility want enkel public
- Interface is ook een type

```
SoundProducer producer=new SoundProducer(); //nope
SoundProducer myCat=new Cat();
```


SoundProducer
makeSound() void
Dog
makeSound() void
Human
makeSound() void
Cat
makeSound() void

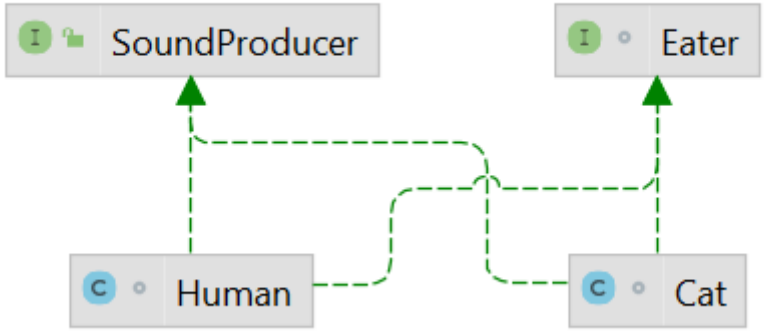
Verschil abstracte class en interface

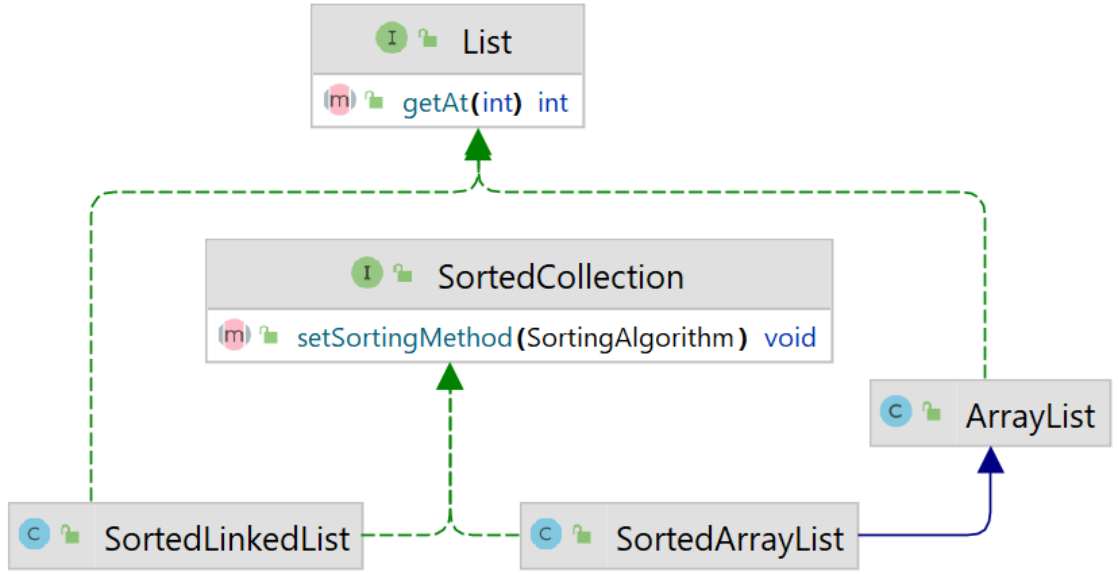
- Een class kan meerdere interfaces implementeren maar slechts 1 andere class overerven.

```
public interface SoundProducer {
    void makeSound();
}
interface Eater {
    void eat();
}

class Human implements SoundProducer,Eater {
    @Override public void makeSound() {
        System.out.println("Hallo");
    }
    @Override public void eat() {
        System.out.println("Eating anything");
    }
}
```

```
class Dog implements SoundProducer, Eater {
    @Override public void makeSound() {
        System.out.println("Woof");
    }

    public void fetch() {
        System.out.println("fetching");
    }

    @Override public void eat() {
        System.out.println("Eating kibbles");
    }
}
```

Voorbeelden multiple interfaces





- Een SortedArrayList gedraagt zich als een List want met getAt(index) kan je het element op de positieve index opvragen.
- Een SortedArrayList gedraagt zich als een SortedCollection want de elementen zijn gesorteerd en met setSortingMethod(SortingAlorithm) kan je bepalen op welke wijze de elementen gesorteed zijn.

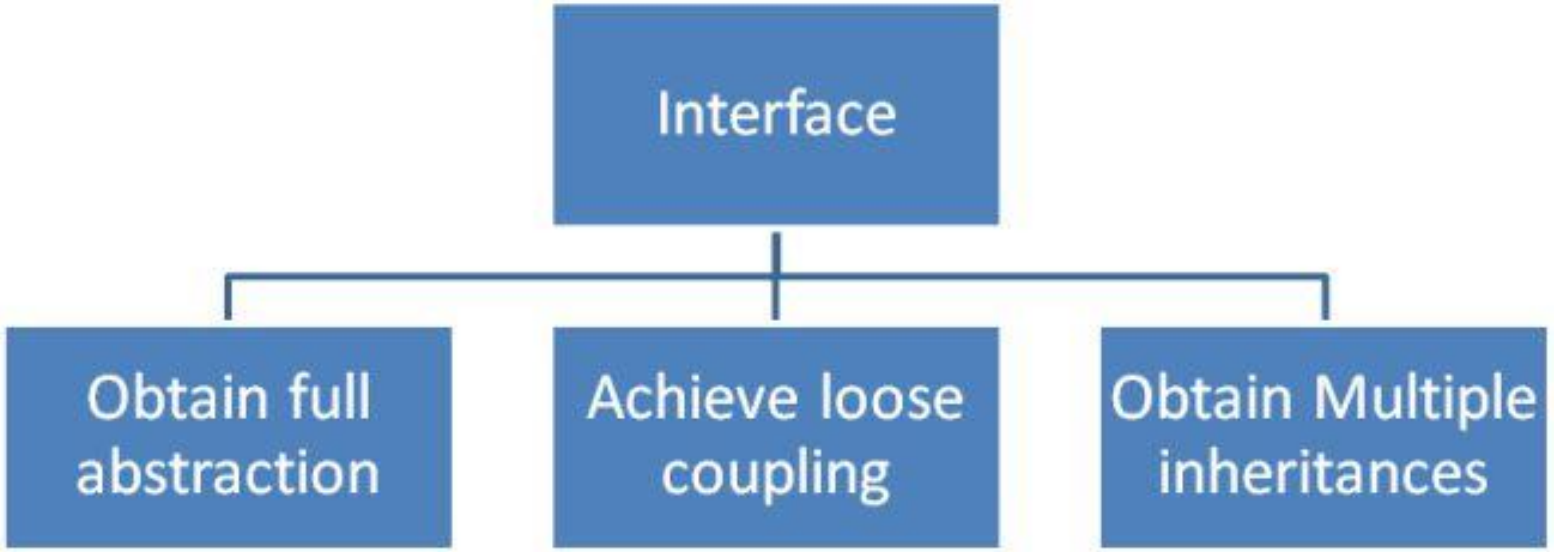
Default implementaties

- Een method in een interface kan een default implementatie voorzien.

```
public interface SoundProducer {
   void makeSound();
}
interface Eater {
   default void eat() {
      System.out.println("Eat whatever is nearby");
   }
}

class Human implements SoundProducer,Eater {
   @Override public void makeSound() {
      System.out.println("Hallo");
   }
}
```

```
class Dog implements SoundProducer, Eater {
   @Override public void makeSound() {
      System.out.println("Woof");
   }

   public void fetch() {
      System.out.println("fetching");
   }

   @Override public void eat() {
      System.out.println("Eating kibbles");
   }
}
```

Waarom interfaces in Java?



Exceptions

Exceptions

- Wat zijn exceptions?
- Is een exception een error?
- Hoe kan je een exception verwerken?

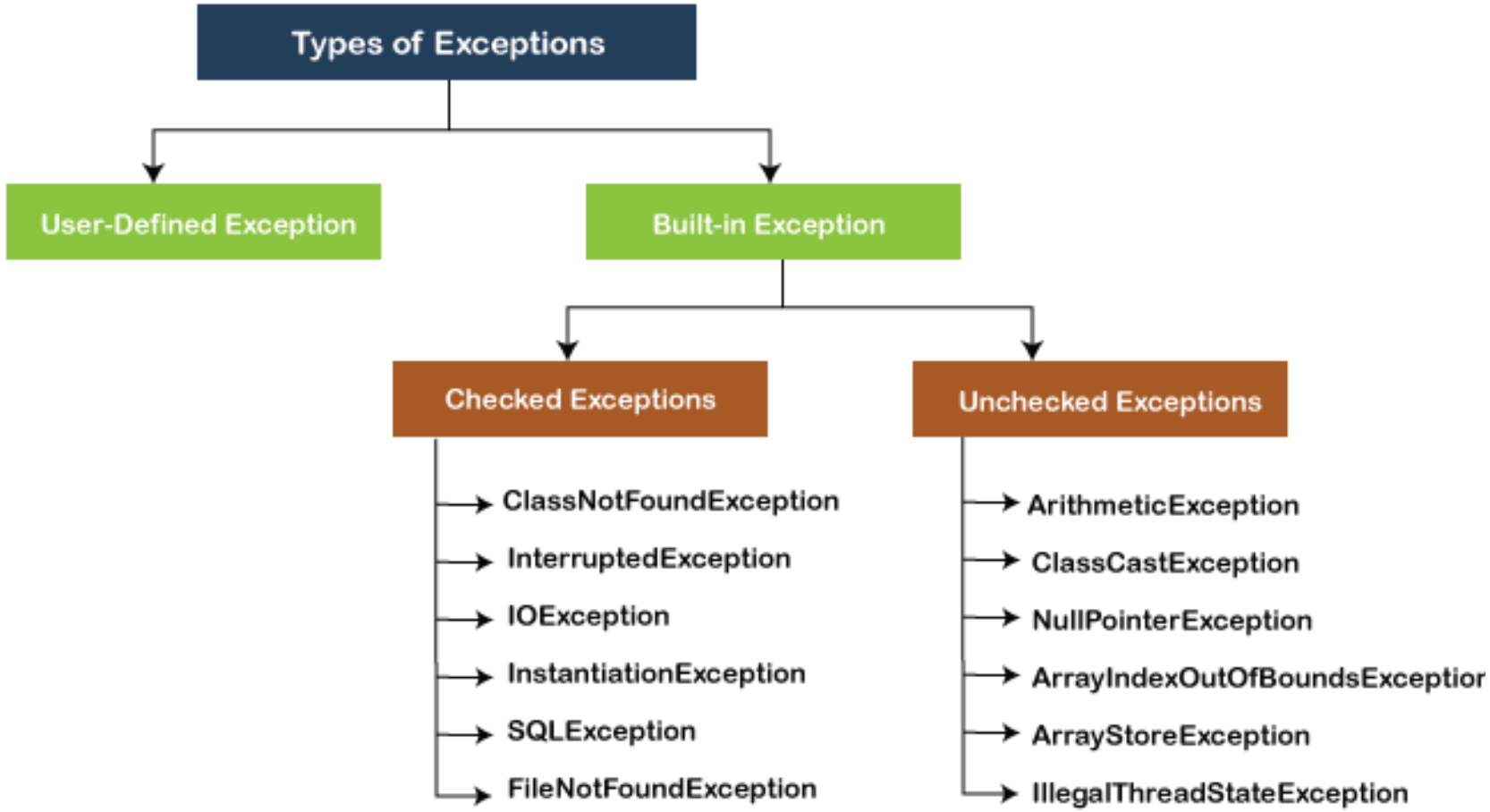
Exceptions

- Ongewenste of onverwacht event
- Verstoord de normale flow van de applicatie
- Kan opgelost worden door de exception of te vangen en te verwerken
  - Try – catch
- Een exception is een object
  - Naam van de exception
  - Beschrijving van het probleem
  - Status van het programma op het moment dat het misging

Errors

- Onherstelbare fouten
  - Out of memory, memory leaks, stack overflows, infinite recursion, …
- Vaak buiten de controle van de programmeur
  - Best geen try-catch gebruiken om het op te lossen

Types of exceptions



Built-in Exceptions

- Aanwezig in de standaard java libraries
- Checked exceptions
  - Compile-time exceptions
- Unchecked exceptions
  - Runtime exceptions

User-Defined Exceptions

- Voor exceptions specifiek voor je applicatie
- Voordelen:
  - Vermijden van voortbouwen van errors
  - Duidelijkere splitsing van programma code en error-handling code
  - Duidelijkere rapportering van fouten
  - Verscheidene types fouten gespecifieerd worden
  - Vermijden dat de applicatie crashed

Hoe worden exceptions verwerkt?

- Een functie werpt (throwed) een exception
  - Functie stopt onmiddellijk
  - Fout wordt doorgegeven naar hoger-gelegen functies
  - Indien ze opgevangen wordt in een try-catch structuur gaat de applicatie verder, de eerste catch die tegengekomen wordt vangt de exception op
  - Indien ze niet opgevangen wordt crasht de applicatie

Exceptions – Oefening – Wat wordt er uitgeprint?

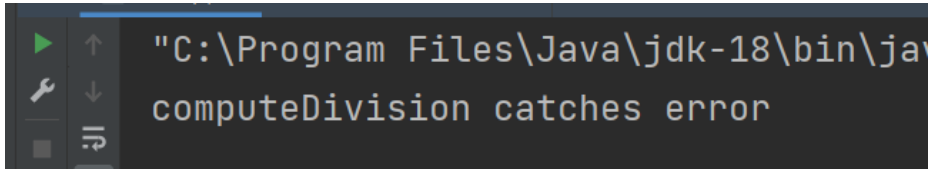
```java
public class Calculator {

    private int divide(int a, int b){
        return a/b;
    }

    public int computeDivision(int a, int b){
        try{
            return divide(a, b);
        } catch (ArithmeticException ex){
            System.out.println("computeDivision catches error");
            return 0;
        }
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        Calculator calculator = new Calculator();

        try {
            calculator.computeDivision( a: 5,  b: 0);
        } catch (ArithmeticException ex){
            System.out.println("Main functie catches the error"
        }

    }
}
```

Exceptions – Oefening – Wat wordt er uitgeprint?

```java
public class Calculator {

    private int divide(int a, int b){
        return a/b;
    }

    public int computeDivision(int a, int b){
        try{
            return divide(a, b);
        } catch (ArithmeticException ex){
            System.out.println("computeDivision catches error");
            return 0;
        }
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        Calculator calculator = new Calculator();

        try {
            calculator.computeDivision( a: 5,  b: 0);
        } catch (ArithmeticException ex){
            System.out.println("Main functie catches the error"
        }

    }
}
```



User-Defined Exceptions - Voorbeeld

```java
public class Person {

    private String name;
    private int age;

    public Person(String name, int age) throws InvalidNameException, InvalidAgeException {

        if(name == null || name.equals("")){
            throw new InvalidNameException();
        } else if(age < 0){
            throw new InvalidAgeException();
        }

        this.name = name;
        this.age = age;
    }

}
```

```java
public class InvalidNameException extends Exception{
}

 public class InvalidAgeException extends Exception{
 }
```

User-Defined Exceptions - Voorbeeld

```java
public class Person {

    private String name;
    private int age;

    public Person(String name, int age) throws InvalidNameException, InvalidAgeException {

        if(name == null || name.equals("")){
            throw new InvalidNameException();
        } else if(age < 0){
            throw new InvalidAgeException();
        }

        this.name = name;
        this.age = age;
    }

}
```

User-Defined Exceptions - Voorbeeld

```java
public class Main {

    public static void main(String[] args) {

        try {
            //Person person = new Person("", 18);
            //Person person = new Person("jens", -18);
            Person person = new Person( name: "",  age: -18);
        } catch (InvalidAgeException e) {
            System.out.println("Catch only invalid age exceptions");
        } catch (Exception e) {
            System.out.println("Catch all exceptions, including invalid name");
        } finally {
            System.out.println("This is optional");
            System.out.println("Something that will happen after try-catch block
        }
    }
}
```

Huiswerk

Geen extra opdracht, maar:

- Verder afwerken van het project
  - Feedback over eerste versie ten laatste op vrijdag 03/06
  - Finale deadline code op 09/06 om 23u59
  - Presentaties op 13/14 juni: planning volgt nog op Toledo

- Indien gewenst: optioneel mondeling examen
  - zelf in te schrijven door dit formulier in te vullen: https://forms.gle/CWCdQWULQbwfrrpf9