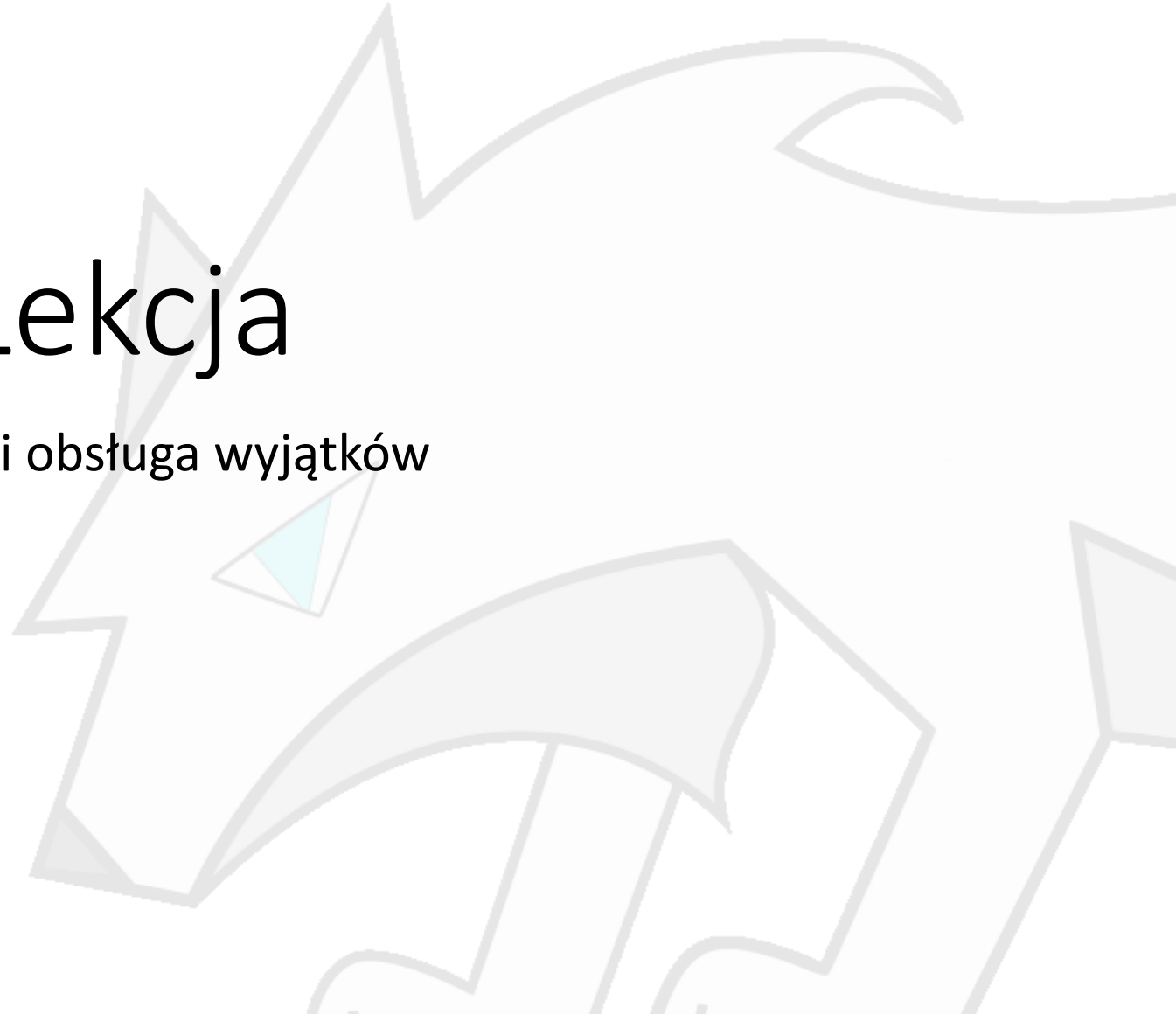


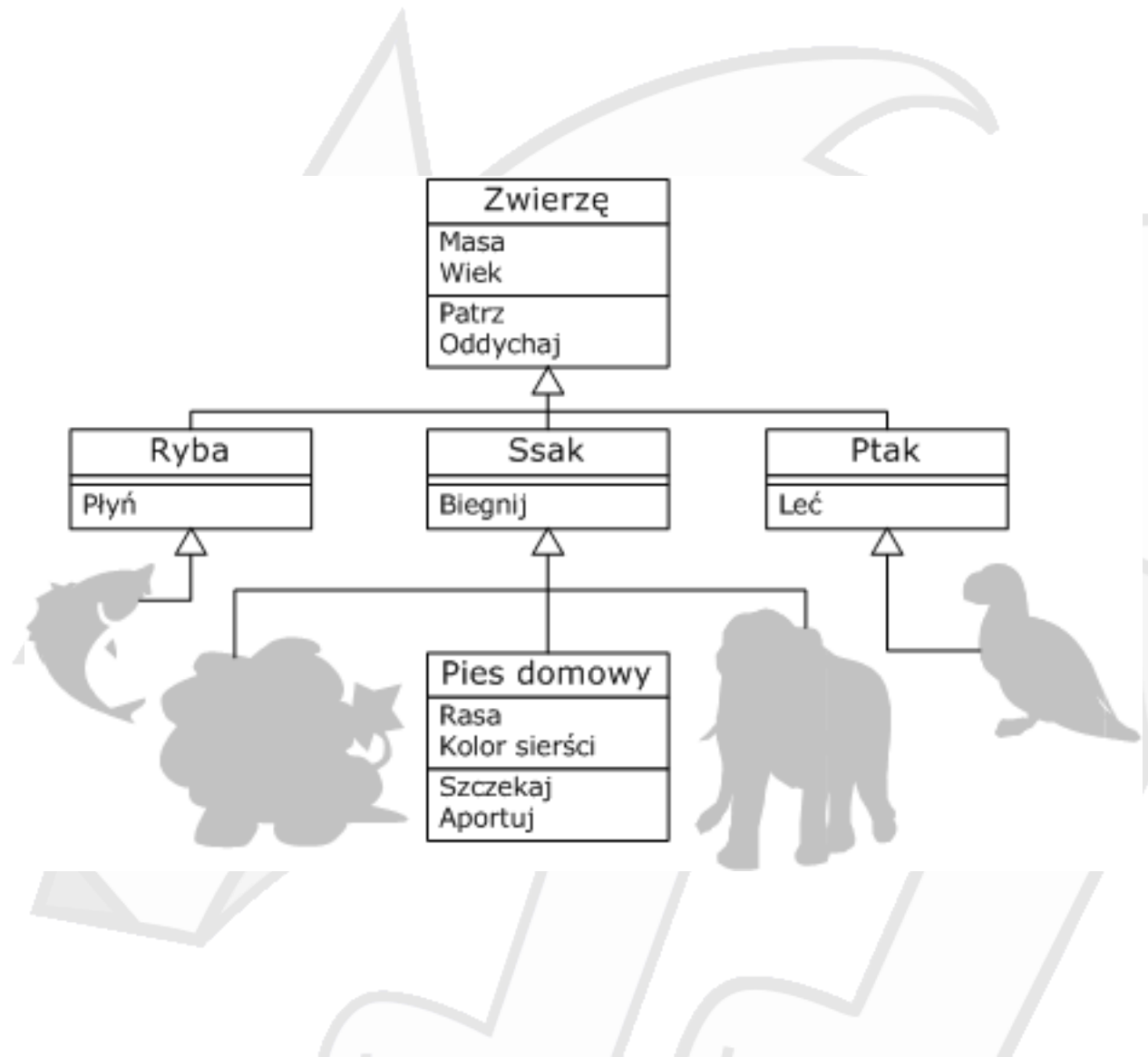
# 9. Lekcja

Dziedziczenie i obsługa wyjątków



# Mechanizm dziedziczenia

mechanizm współdzielenia funkcjonalności między klasami. Klasa może dziedziczyć po innej klasie, co oznacza, że oprócz swoich własnych atrybutów oraz zachowań, uzyskuje także te pochodzące z klasy, z której dziedziczy. Klasa dziedzicząca jest nazywana **klasą pochodną lub potomną** (w j. angielskim: subclass lub derived class), zaś klasa, z której następuje dziedziczenie — **klasą bazową**



# Słówka kluczowe

- extends – słówko kluczowe mówiące że klasa następująca po nim jest klasą jest bazową do klasy którą słówko to poprzedza.

```
public class Fish extends Animal {
```

- super – słówko kluczowe mówiące o tym aby został wykorzystany do zbudowania obiektu konstruktor z klasy bazowej zgodny z podaną listą parametrów.

```
public Fish(String name, String type, int weight, boolean freshwater) {  
    super(name, type, weight);  
    this.freshwater = freshwater;  
}
```

```
public class Animal {
```

```
    private String name;  
    private String type;  
    private int weight;
```

```
    public Animal(String name, String type, int weight) {  
        this.name = name;  
        this.type = type;  
        this.weight = weight;  
    }
```

```
    public void breathing(){  
        System.out.println("Breath breath");  
    }
```

```
    public String getName() {  
        return name;  
    }
```

```
    public String getType() {  
        return type;  
    }
```

```
    public int getWeight() {  
        return weight;  
    }
```

```
}
```

```
public class Fish extends Animal {
```

```
    private boolean freshwater;
```

```
    public Fish(String name, String type, int weight, boolean freshwater) {  
        super(name, type, weight);  
        this.freshwater = freshwater;  
    }
```

```
    public void swimming() {  
        System.out.println("Plum plum!");  
    }
```

```
    public boolean isFreshwater() {  
        return freshwater;  
    }
```

```
}
```

FileEditViewNavigateCodeAnalyzeRefactorBuildRunToolsVCSWindowHelp

Demo [C:\Users\dobne\IdeaProjects\Demo] - Main.java

Demo > src > com > company > Main

Project

Demo C:\Users\dobne\IdeaProjects\Demo

.idea

out

src

com.company

Animal

Fish

Main

Demo.iml

External Libraries

Scratches and Consoles

Main.java x Fish.java x Animal.java x

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

```
Animal animal = new Animal( name: "Some animal", type: "Undefined", weight: 100);
Fish carp = new Fish( name: "Carp", type: "Fish", weight: 1, freshwater: true);

Animal maybeFish = carp;
// Fish alwaysFish = animal;

System.out.print("Animal breathing ");
animal.breathing();

System.out.print("Fish breathing ");
carp.breathing();

System.out.print("Fish swimming ");
carp.swimming();
// animal.swimming();

System.out.print("Maybe fish swimming ");
((Fish)maybeFish).swimming();

System.out.println("Fish name " + carp.getName()
    + " and he a " + carp.getType()
    + " his weight is " + carp.getWeight()
    + " and he is " + (carp.isFreshwater() ? "freshwater" : "not freshwater"));
```

3

1

^

v

Run: Main x

Animal breathing Breath breath

Fish breathing Breath breath

Fish swimming Plum plum!

Maybe fish swimming Plum plum!

Fish name Carp and he a Fish his weight is 1 and he is freshwater

4: Run

TODO

6: Problems

Terminal

Build

Event Log

Build completed successfully in 11 s 282 ms (a minute ago)

33:1 CRLF UTF-8 4 spaces

# Co to jest wyjątek ?



Mechanizm sterowania przepływu używany w procesorach oraz współczesnych językach programowania do obsługi zdarzeń wyjątkowych, a w szczególności błędów, których wystąpienie zmienia prawidłowy przebieg wykonywania programu.

# Co to jest obsługa wyjątku ?

Jeżeli podczas wykonywania naszego kodu wystąpi niespodziewany błąd, powinniśmy być na to przygotowanie i wychwycić taką sytuację. Inaczej gdy wyjątek jest rzucony „throw” musimy spróbować „try” go złapać „catch”

```
try {  
    // operations which can return exception  
} catch (Exception e){  
    // if operations will throw exception, here we catch it  
}
```

# Słówka kluczowe

```
public void metodePossibleToThrowException() throws IllegalAccessException {  
    String nullString = null;  
    try {  
        nullString.isEmpty();  
    } catch (NullPointerException e) {  
        System.out.println("String is null. Can not be empty.");  
    } catch (ArithmeticException e) {  
        System.out.println("Oh no.. We have some arthmetical problems");  
    } finally {  
        throw new IllegalAccessException("Unexceptional problem. We must close the system!");  
    }  
}
```

- Try – w tej sekcji spodziewamy się wyrzucenia wyjątku
- Catch – przechwytuje sterowanie gdy napotka wyjątek danego typu.
- Throw – powoduje manualne wyrzucenie wyjątku.
- Throws – informuje, że metoda może zwrócić wyjątek.
- Finally – niezależnie od rzucenia lub przechwycenia wyjątku ta część kodu zawsze się wykona




 Main
 








Ⓢ Main.java ✕

## Scratches and Consoles

4 ^ v



Event Log

26:2 CRLF UTF-8 4 spaces 🔒

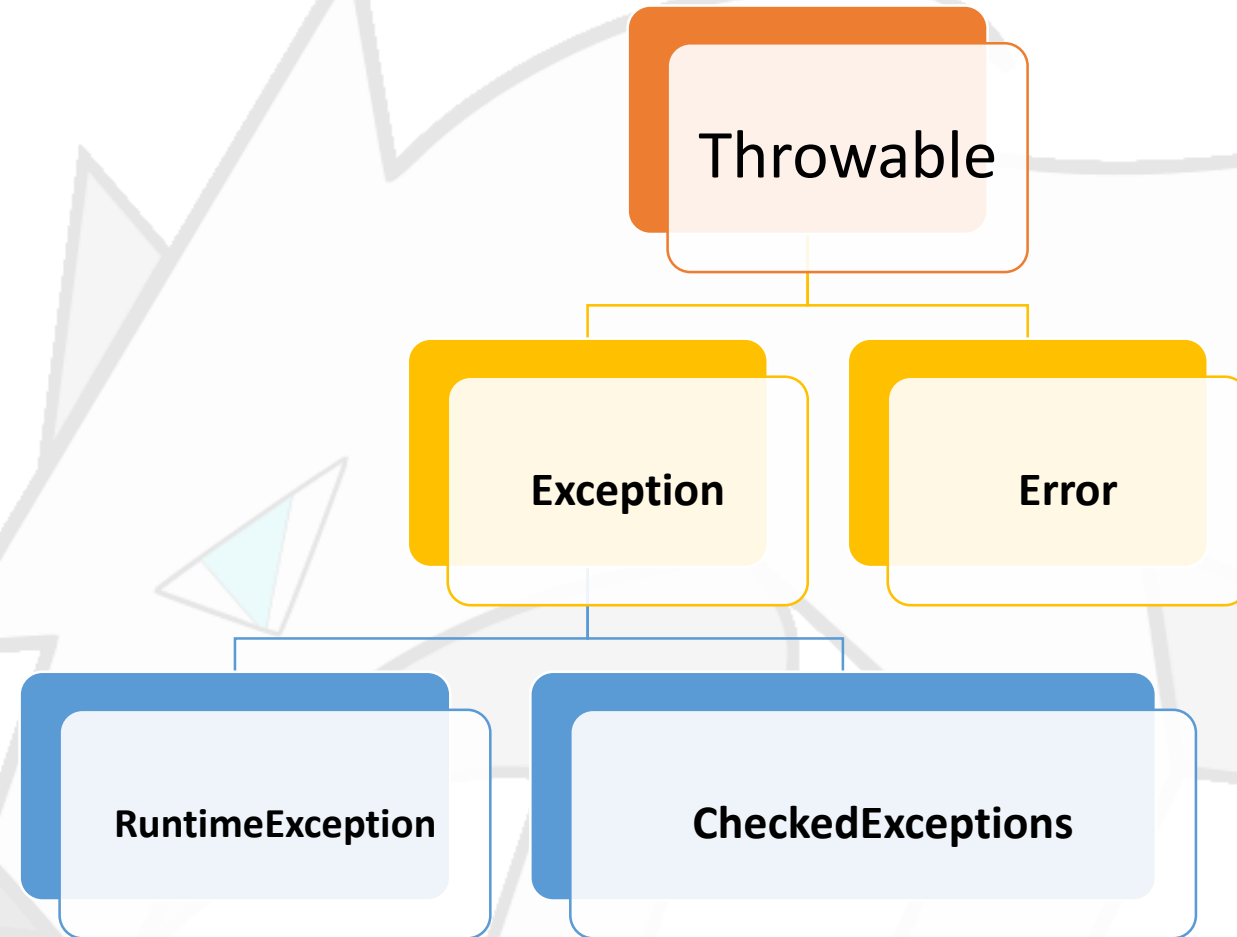
# Z czego składa się exception ?

- message – zawiera informację o powodzie wyrzucenia błędu.
- stackTrace – zapis stosu wywołań, reprezentujący obecny stan programu. Zawiera informację o miejscu gdzie został wyrzucony wyjątek.

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 100 out of bounds for length 5  
    at com.company.Main.main(Main.java:9)
```

# Error vs Exception

- Error – obiekty rzucane automatycznie przez JVM, w momencie gdy program natrafi na krytyczny błąd w środowisku wykonawczym, którego nie można obsłużyć i z którego nie można programu już odratować.
- Exception – najczęściej dotyczą sytuacji, w której jest duża szansa, że błąd może wystąpić na skutek jakiejś zewnętrznej przyczyny – np. w systemie gospodarza zabraknie odpowiedniego pliku lub nie uda się połączyć z serwerem. Program musi być koniecznie gotowy na taką ewentualność.



# RuntimeException

```
public void metodePossibleToThrowException() {  
    //RuntimeException  
    throw new NullPointerException();  
}
```

```
public void metodePossibleToThrowException() throws NullPointerException {  
    //RuntimeException  
    throw new NullPointerException();  
}
```

```
public void metodePossibleToThrowException() {  
    //RuntimeException  
    try {  
        throw new NullPointerException();  
    } catch (NullPointerException e) {  
        System.out.println("CAUGHT");  
    }  
}
```

- Nie musi być przechwytywany ale może być. Jeśli nie zostanie przechwycony przechwyci go klasa main().
- Nie musi ale może być przerzucany dalej.
- Dokładnie te same zasady dotyczą Errorów z tą różnicą, że można ale nie powinno się ich przechwytywać.

# CheckedExceptions

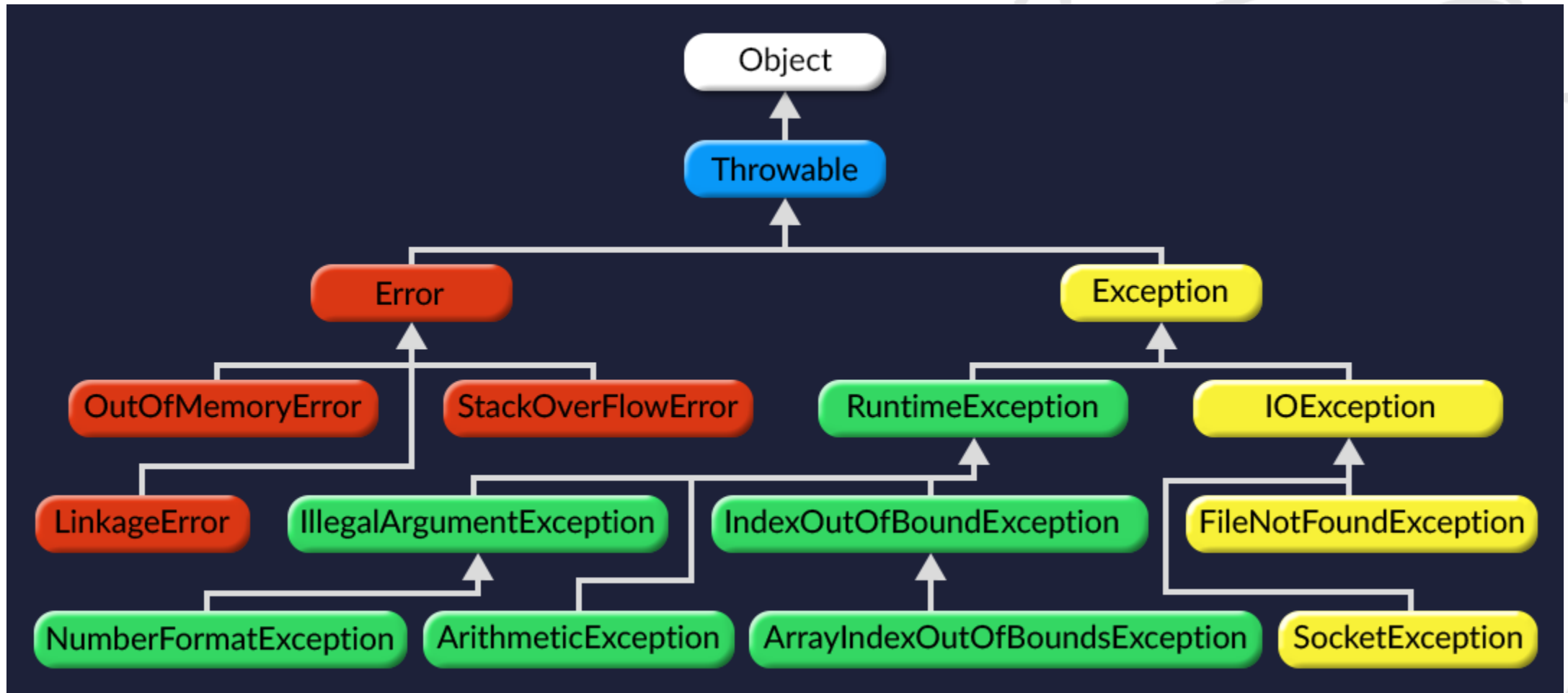
```
public void metodePossibleToThrowException() {  
    //Exception  
    throw new FileNotFoundException();  
}
```

```
public void metodePossibleToThrowException() throws FileNotFoundException {  
    //Exception  
    throw new FileNotFoundException();  
}
```

```
public void metodePossibleToThrowException() {  
    //Exception  
    try {  
        throw new FileNotFoundException();  
    } catch (FileNotFoundException e) {  
        System.out.println("CAUGHT");  
    }  
}
```

- Pozostawienie go bez bloku catch albo przerzucenia powoduje błąd kompilacji
- **Musi** zostać albo przechwycony albo przekazany dalej.

# Hierarchia wyjątków



# Dziedziczenie wyjątków i hierarchia przechwytywania

W taki sam sposób jak dziedziczymy klasy możemy też tworzyć własne wyjątki.

Jeżeli mamy więcej bloków „catch” wyjątek zostanie przechwycony w tym do którego pasuje jako pierwsze. Niezależnie od tego czy wyjątek zostanie przechwycony czy nie, jeśli jakikolwiek wyjątek zostanie rzucony w bloku try, blok finally i tak się wykona.

```
public class MySpecialException extends Exception {  
  
    @Override  
    public String getMessage() {  
        return "Panic message";  
    }  
  
}
```

```
try {  
    throw new MySpecialException();  
} catch (MySpecialException e) {  
    System.out.println(e.getMessage());  
} catch (NullPointerException e) {  
    System.out.println("Oh no... Null");  
} catch (Exception e) {  
    System.out.println("Some exception...");  
} finally {  
    System.out.println("Finally to!");  
}
```

I wyjątkowo teraz zadania ;)

