

# Własne klasy wyjątków

## Czego się dowiesz

- Jak definiować własne typy wyjątków,
- po co i kiedy to robić.

Czasami mogą zdarzać się sytuacje, w których chcielibyśmy poinformować, że nasza metoda może generować jakiś wyjątek, jednak w standardowej hierarchii klas Javy nie możemy znaleźć takiego, który by najlepiej odpowiadał naszej sytuacji. Na szczęście dzięki dziedziczeniu możemy rozszerzyć dowolną z klas wyjątków i definiować własne typy błędów.

W praktyce warto pamiętać o tym, że wszystkie wyjątki dziedziczące po klasie `RuntimeException` nie muszą być obsługiwane, natomiast jeśli stworzymy klasę dziedziczącą po `Exception`, to wyjątek taki będzie musiał być obsługowany, co pokażemy na przykładzie. Dobrą praktyką jest nazywanie swoich klas w taki sposób, aby zawierały one na końcu słowo *Exception* - dzięki trzymaniu się takiej konwencji wskazujemy jasno jej zastosowanie.

Stworzenie klasy wyjątku jest dosyć proste i najczęściej sprowadzi się jedynie do dodania opcjonalnych konstruktorów.

Jako przykład stwórzmy klasę `School`, a w niej tablicę typu `Student`, gdzie każdy student ma imię, nazwisko i id (np. numer legitymacji). W klasie `School` dodajmy metodę `add()` wprowadzającą nowego studenta do systemu oraz `find()`, która jako argument przyjmuje `String` z imieniem i nazwiskiem i zwraca pierwszego `Studenta` znalezionego w tablicy. Ponieważ w przypadku metody `add()` może okazać się, że nie ma już więcej miejsca w tablicy - utwórzmy swój wyjątek `NoMoreSpaceException`, a metoda `find()` może nie znaleźć żadnego elementu i wtedy zamiast zwracać wartość `null` może zwrócić wyjątek `NoElementFoundException`. Ponieważ nie potrzebujemy w klasach wyjątków jakichś specjalnych funkcjonalności wystarczy jeśli zdefiniujemy jeden konstruktor przyjmujący obiekt `String` z dowolnym komunikatem, który później będzie można odczytać przy łapaniu obiektu wyjątku.

*Student.java*

```
class Student {
    private int studentId;
    private String firstName;
    private String lastName;

    public Student(int studentId, String firstName, String lastName) {
        this.studentId = studentId;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public int getStudentId() {
        return studentId;
    }

    public void setStudentId(int studentId) {
        this.studentId = studentId;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    @Override
    public String toString() {
        return studentId + " " + firstName + " " + lastName;
    }
}
```

*NoMoreSpaceException.java*

```
class NoMoreSpaceException extends Exception {
    public NoMoreSpaceException(String message) {
        super(message);
    }
}
```

*NoElementFoundException.java*

```
class NoElementFoundException extends Exception {
    public NoElementFoundException(String message) {
        super(message);
    }
}
```

Obie klasy wyjątków rozszerzają klasę `Exception`, ponieważ chcemy, aby wymusić ich obsługę. Definiujemy jeden konstruktor przyjmujący dowolny komunikat, który będzie mógł być później wyświetlony. Komunikat będzie zapisany w polu prywatnym z klasy `Exception`. Możemy go zapisać korzystając z instrukcji `super()`, która pozwala wywołać konstruktor klasy, po której dziedziczymy.

*School.java*

```
class School {
    private Student[] students;
    private int studentsNumber;

    public School(int studentsNumber) {
        students = new Student[studentsNumber];
    }

    public void add(Student s) throws NoMoreSpaceException {
        if (studentsNumber >= students.length) {
            throw new NoMoreSpaceException("Brak miejsca w School " +
students.length);
        } else {
            students[studentsNumber] = s;
            studentsNumber++;
        }
    }

    public Student find(String firstName, String lastName) throws
NoElementException {
        int index = 0;
        while (index < students.length) {
            if (students[index].getFirstName().equals(firstName) &&
students[index].getLastName().equals(lastName)) {
                return students[index];
            } else {
                index++;
            }
        }

        throw new NoElementException("Nie znaleziono elementu " + firstName
+ " " + lastName);
    }
}
```

Tablica `students` przechowuje studentów w szkole, a zmienna `studentsNumber` mówi o aktualnej liczby studentów. Metoda `add()` może wygenerować wyjątek kontrolowany `NoMoreSpaceException` w przypadku, gdy już zabraknie miejsca w tablicy. Metoda `find()` służy do wyszukania studenta o podanym imieniu i nazwisku. Przeszukujemy w niej tablicę i jeżeli trafimy na pierwszy element, którego imię i nazwisko (pola *firstName* i *lastName*) zgadzają się z argumentami metody, to zwracamy znaleziony obiekt.

W przypadku, gdy nie znajdziemy żadnego studenta o wskazanym imieniu i nazwisku, to mamy kilka możliwości. Moglibyśmy np. zwrócić wartość null, jednak to wymagałoby późniejszego sprawdzania na każdym dalszym kroku aplikacji, czy zwrócony student jest nullem, czy nie. My decydujemy się więc na inne rozwiązanie, w którym w przypadku braku studenta rzucamy wyjątek.

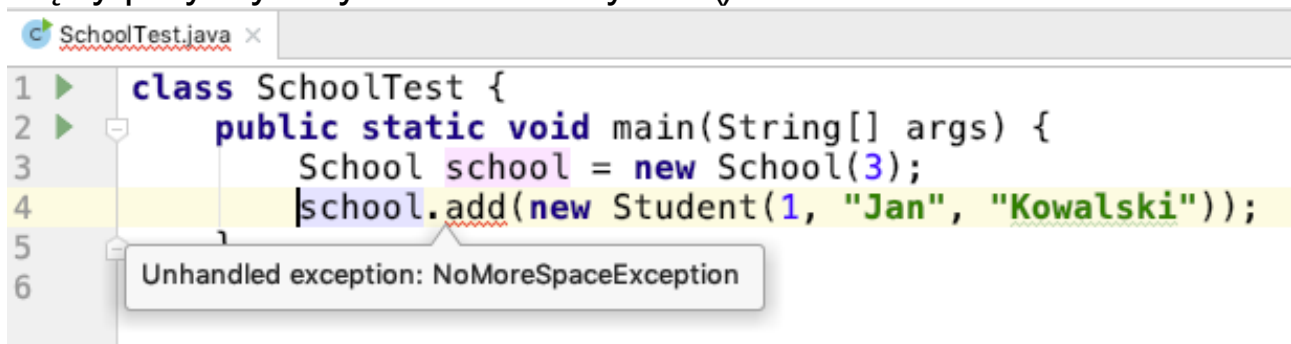
Ponieważ nasze wyjątki dziedziczą po klasie Exception, więc są wyjątkami kontrolowanymi, to jeśli je rzucamy, musimy jednocześnie zadeklarować je używając słowa throws w sygnaturze metody. Przy wyjątkach niekontrolowanych jest to opcjonalne i najczęściej się tego nie zapisuje.

W celu przetestowania działania klasy stwórzmy klasę SchoolTest.

*SchoolTest.java*

```
class SchoolTest {  
    public static void main(String[] args) {  
        School school = new School(3);  
        school.add(new Student(1, "Jan", "Kowalski"));  
    }  
}
```

Po jej zapisaniu w takiej postaci środowisko od razu podkreśli błędy przy wywoływaniu metody add():



Ponieważ nasze wyjątki rozszerzają bezpośrednio klasę Exception, to znaczy, że są kontrolowane, a tym samym muszą zostać obsłużone. Jeżeli nasze klasy wyjątków, czyli NoMoreSpaceException i NoElementFoundException dziedziczyłyby po klasie RuntimeException, to w klasie SchoolTest nie pojawiłby się żaden błąd.

Po wciśnięciu kombinacji klawiszy Alt+Enter w IntelliJ lub Ctrl + 1 w eclipse otrzymujemy podpowiedź rozwiązania problemu:

Widzimy, że możemy dodać deklarację throws do sygnatury metody i nie przejmować się obsługą wyjątku. Z tego podejścia skorzystamy wtedy, kiedy jesteśmy w takim miejscu kodu, w którym nie powinny znajdować się instrukcje odpowiedzialne za komunikację z użytkownikiem. W naszym przypadku nad klasą SchoolTest nie ma już nic więcej, dodanie słowa throws do metody main() oznaczałoby zignorowanie wyjątku, a w efekcie przerwanie działania aplikacji z wyświetleniem stacktrace w konsoli.

Problematiczny fragment kodu lepiej jest ująć w blok try catch (opcja surround with try catch w IntelliJ) i poinformować użytkownika, że coś poszło nie tak. Analogicznie postąpimy z metodą find().

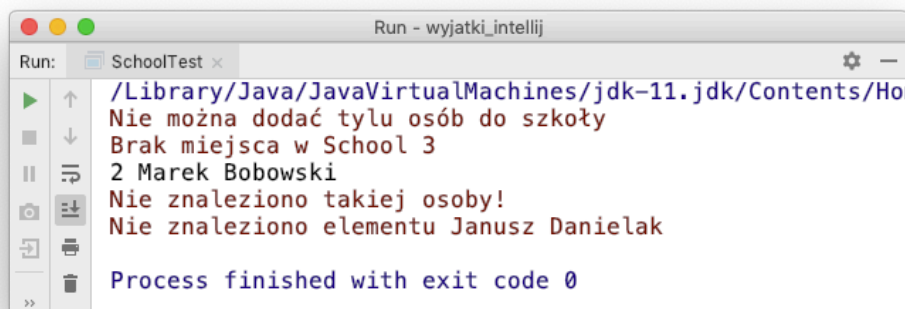
*SchoolTest.java*

```
class SchoolTest {
    public static void main(String[] args) {
        School school = new School(3);
        try {
            school.add(new Student(1, "Jan", "Kowalski"));
            school.add(new Student(2, "Marek", "Bobowski"));
            school.add(new Student(3, "Karol", "Kwiatkowski"));
            school.add(new Student(4, "Ania", "Marciniak"));
        } catch (NoMoreSpaceException e) {
            System.err.println("Nie można dodać tylu osób do szkoły");
            System.err.println(e.getMessage());
        }

        try {
            System.out.println(school.find("Marek", "Bobowski"));
            System.out.println(school.find("Janusz", "Danielak"));
        } catch (NoElementException e) {
            System.err.println("Nie znaleziono takiej osoby!");
            System.err.println(e.getMessage());
        }
    }
}
```

Stworzyliśmy obiekt *School*, który może przechowywać informację o maksymalnie 3 studentach. Przy próbie dodania czwartego z nich rzucony będzie wyjątek *NoMoreSpaceException*, który przechwytujemy w bloku *catch* i obsługujemy. Podobnie w drugim bloku *try* wyszukujemy *Marka Bobowskiego*, który istnieje w naszej bazie, ale przy wyszukiwaniu studenta *Janusz Danielak* otrzymamy wyjątek *NoElementFoundException*, który też przechwytujemy i obsługujemy.

Po uruchomieniu programu możesz (ale nie musisz) zobaczyć wynik w różnej kolejności. Jest to spowodowane tym, że strumienie *System.out* i *System.err* nie są ze sobą zsynchronizowane.



```
Run: SchoolTest x
/Library/Java/JavaVirtualMachines/jdk-11.jdk/Contents/Home
Nie można dodać tylu osób do szkoły
Brak miejsca w School 3
2 Marek Bobowski
Nie znaleziono takiej osoby!
Nie znaleziono elementu Janusz Danielak
Process finished with exit code 0
```

Własne klasy wyjątków definiuj tylko wtedy, kiedy wśród standardowych klas nie znajdziesz odpowiednich do swojego problemu. Przykładowo definiowanie wyjątku typu *TooBigIndexException* w odniesieniu do zbyt dużego indeksu tablicy będzie złą praktyką, ponieważ istnieje *ArrayIndexOutOfBoundsException*. Własne wyjątki będą przydatne szczególnie wtedy, kiedy mamy w niej do rozróżnienia wiele nietypowych sytuacji.

Java jest jedynym językiem programowania, w którym wprowadzono podział na wyjątki kontrolowane i niekontrolowane. Aktualnie praktycznie wszystkie wyjątki, które definiuje się w aplikacjach to wyjątki niekontrolowane (dziedziczące po *RuntimeException*), często wyjątki kontrolowane uważane są za błąd przy projektowaniu języka Java i raczej staraj się ich unikać w swoich aplikacjach.