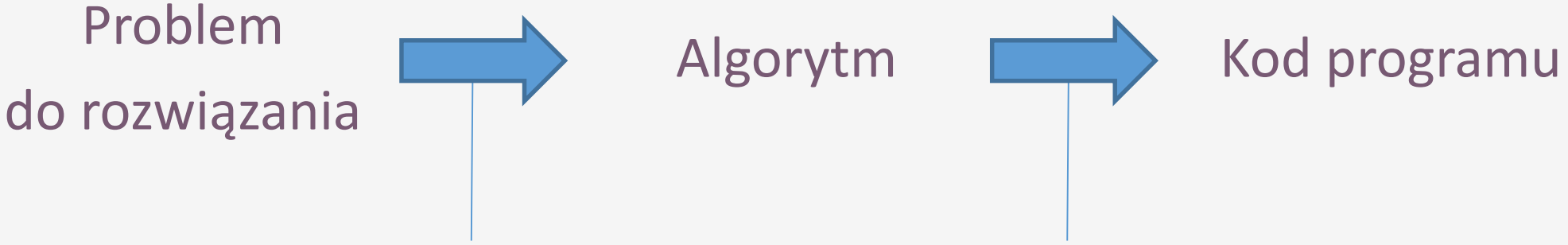
Wprowadzenie do języka JAVA

Marek Sontag
mareksontag@gmail.com

AGENDA

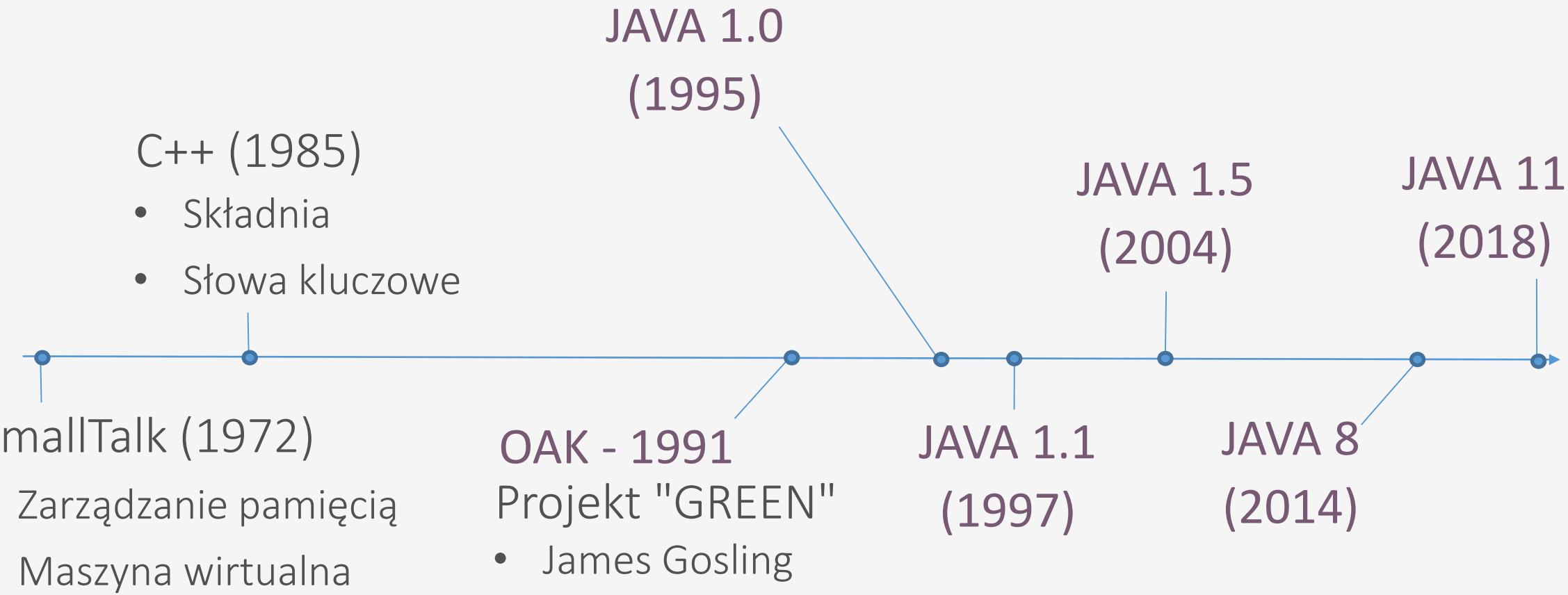
1. Przedstawmy się!
2. Czym jest programowanie, rys historyczny języka JAVA
3. Założenia języka
4. Pierwszy program
5. IDE
6. Dane w programach: zmienne, stałe
7. Klasa – pole, metoda, konstruktor
8. Pakiet, modyfikatory dostępu, import
9. Dziedziczenie
10. Maven
11. Testowanie
12. Typy
13. Operacje na danych: operatory

CZYM JEST PROGRAMOWANIE?



Autor: Marek Sontag

RYS HISTORYCZNY



ZAŁOŻENIA JĘZYKA JAVA

OBIEKTOWOŚĆ

Obiekty, przedmioty ze świata realnego jako obiekty wirtualne

BEZPIECZEŃSTWO

(ang. safety, ang. security)

Autor: Marek Sontag

ZAŁOŻENIA JĘZYKA JAVA

PRZENOŚNOŚĆ

Raz napisany program uruchamialny na różnych platformach (np. Windows, Linux, macOS)

NIEZALEŻNOŚĆ OD ARCHITEKTURY

Komputery PC, urządzenia mobilne, systemy embedded

Autor: Marek Sontag

ZAŁOŻENIA JĘZYKA JAVA

WIELOWĄTKOWOŚĆ

Równoczesne wykonywanie różnych zadań

WYDAJNOŚĆ

Programy wykonywane efektywnie

Autor: Marek Sontag

PIERWSZY PROGRAM

"Hello, world!"- jak?

1. Środowisko: JDK, JAVA_HOME
2. Napisanie kodu
3. Kompilacja
4. Uruchomienie

JDK – Java Development Kit

JRE – Java Runtime Environment

Autor: Marek Sontag

ZADANIA

1. Zainstaluj pakiet JDK dla Javy 8
2. Sprawdź wersję Javy
3. Utwórz klasę Runner.java, która wypisuje: `Hello, World!`
4. Skompiluj klasę Runner.java
5. Uruchom klasę `Runner`
6. Stwórz archiwum JAR dla swojej aplikacji
7. Uruchom stworzone archiwum
8. Stwórz plik manifest ze wskazaniem klasy startowej `Runner`
9. Stwórz archiwum JAR z własnym manifestem
10. Uruchom stworzone archiwum

Autor: Marek Sontag

IDE – Integrated Development Environment

Zapewnia środowisko do tworzenia, testowania i uruchamiania programów

- IntelliJ
- NetBeans
- Eclipse
- inne

Autor: Marek Sontag

IDE – Integrated Development Environment

IntelliJ – obecnie najpopularniejsze narzędzie wśród programistów Java

- Wersje:
  - Community (darmowa, dla nas wystarczająca)
  - Ultimate (płatna, rozszerzona funkcjonalność)
- Metadane (.idea, nazwa-projektu.iml)
- Pluginy
- Skróty
- I wiele innych…

Autor: Marek Sontag

ZADANIA

1. Pobrać i zainstalować IntelliJ
2. Wydrukować kartkę ze skrótami (w domu)
3. Stworzyć nowy projekt Java korzystając z szablonu Java Hello World
4. Sprawdzić czy dodał się plik nazwa-projektu.iml oraz folder .idea.
5. Uruchomić aplikację z poziomu IntelliJ
6. Dodać plugin "key promoter"

Autor: Marek Sontag

DANE W PROGRAMACH

Jak zapisać i używać danych w programach?

- Zmienne `int zmienna = 1;`
- Stałe `final double PI = 3.14159;`

Autor: Marek Sontag

DANE W PROGRAMACH - ZMIENNE

DEKLARACJA

```
int zmienna;
```

Autor: Marek Sontag

DANE W PROGRAMACH - ZMIENNE

DEKLARACJA

typ

```
int zmienna;
```

nazwa

Autor: Marek Sontag

DANE W PROGRAMACH - ZMIENNE

DEKLARACJA Z INICJALIZACJĄ

```
int zmienna = 123;
```

Autor: Marek Sontag

DANE W PROGRAMACH - ZMIENNE

DEKLARACJA Z INICJALIZACJĄ

oparator przypisania

```
int zmienna = 123;
```

wartość

DANE W PROGRAMACH - ZMIENNE

NADPISANIE WARTOŚCI

```
int zmienna = 123;
...
zmienna = 234;
```

od tej linii `zmienna` ma nową wartośc (234)

Autor: Marek Sontag

DANE W PROGRAMACH - ZMIENNE

PRZYPISANIE WARTOŚCI JEDNEJ ZMIENNEJ DO DRUGIEJ ZMIENNEJ

```
int zmienna1 = 1;
int zmienna2 = zmienna1;
```

← `zmienna2` ma wartość 1

Autor: Marek Sontag

DANE W PROGRAMACH - STAŁE

DEKLARACJA

typ

```
final double PI = 3.14;
```

nazwa

Autor: Marek Sontag

DANE W PROGRAMACH - STAŁE

DEKLARACJA

```
final double PI = 3.14;
```

słowo kluczowe final

DANE W PROGRAMACH - STAŁE

DEKLARACJA

```
final double PI = 3.14;
```

słowo kluczowe final

Stała to inaczej zmienna finalna

Autor: Marek Sontag

DANE W PROGRAMACH - STAŁE

ZMIENNA FINALNA = TYLKO RAZ PRZYPISUJEMY WARTOŚĆ

```
final double PI = 3.14;
PI = 3.14159;
```

Błąd! Nie można zmienić wartości stałej!

ZADANIA

1. W metodzie `main` z poprzedniego zadania dodaj i zainicjuj wybranymi przez Ciebie wartościami:
    1. Stałą `String BRAND`.
    2. Zmienną `String model`.
    3. Zmienną `int maxSpeed`.
2. Wypisz je na standardowe wyjście (System.out.println). Uruchom program.
3. W dalszej części metody zmodyfikuj zmienną `model` oraz `maxSpeed` i ponownie wypisz jak w punkcie 2.
4. Spróbuj zmienić wartość (już zainicjowanej) stałej `BRAND`. Co podpowiada IDE?

KLASA – POLE, METODA, KONSTRUKTOR

KLASA

- Podstawowy element programowania obiektowego
- Reprezentacja rzeczywistego obiektu w programie

Autor: Marek Sontag

KLASA – POLE, METODA, KONSTRUKTOR

KLASA – najprostsza definicja

```
class NazwaKlasy { }
```

słowo kluczowe class

ciało klasy (tu puste)

Autor: Marek Sontag

KLASA – POLE, METODA, KONSTRUKTOR

KLASA – POLE

```
class Car {
    String model;
}
```

Pole klasy

Autor: Marek Sontag

KLASA - KONSTRUKTOR

Wywołanie konstruktora

```
Car mojSamochod = new Car();
```

operator new

Autor: Marek Sontag

KLASA – POLE, METODA, KONSTRUKTOR

KLASA A OBIEKT KLASY (INSTANCJA)

```
class Car {
    String model;
}
public static void main(String[] args) {
    Car mojSamochod = new Car();
}
```

Klasa

obiekt klasy Car (instancja)

Instrukcja tworzenia obiektu (wywołanie konstruktora)

Autor: Marek Sontag

KLASA – POLE, METODA, KONSTRUKTOR

KLASA - KONSTRUKTOR

```
class Car {
      String model;
      Car() {
      }
}
```

definicja konstruktora

KLASA – POLE, METODA, KONSTRUKTOR

KLASA - KONSTRUKTOR

```
class Car {
     String model;
     Car() {
          model = „Forester”;
     }
}
```

Autor: Marek Sontag

KLASA – POLE, METODA, KONSTRUKTOR

KLASA - KONSTRUKTOR

```
class Car {
     String model;
     Car() {
          this.model = „Forester”;
     }
}
```

operator this

KLASA – POLE, METODA, KONSTRUKTOR

KLASA – KONSTRUKTOR ARGUMENTOWY

```
class Car {
    String model;
    Car(String modelArg) {
        this.model = modelArg;
    }
}
```

Autor: Marek Sontag

KLASA – POLE, METODA, KONSTRUKTOR

KLASA – KONSTRUKTOR ARGUMENTOWY

```
class Car {
    String model;
    Car(String model) {
        this.model = model;
    }
}
```

Autor: Marek Sontag

KLASA – POLE, METODA, KONSTRUKTOR

KLASA – METODA

```
class Car {
    String model;
    void printModel() {
        System.out.println(model);
    }
}
```

metoda

Autor: Marek Sontag

KLASA – POLE, METODA, KONSTRUKTOR

KLASA – METODA

```
class Car {
    String model;
    String getModel() {
        return model;
    }
}
```

Typ zwracany

KLASA – POLE, METODA, KONSTRUKTOR

KLASA – METODA

```
class Car {
    String model;
    void setModel(String model) {
        this.model = model;
    }
}
```

metoda wykonuje operacje, nie zwraca wartości

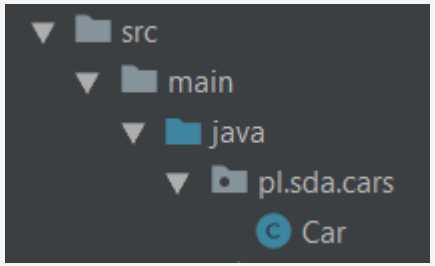
ZADANIA

1. Dodaj do projektu z poprzedniego zadania klasę Car zawierającą:
    1. Pole `String model`.
    2. Konstruktor jednoargumentowy ustawiający pole `model`.
    3. Metodę `printCar()` wyświetlającą model samochodu.
2. W metodzie `main` utwórz dwa obiekty `Car` (rożne modele).
3. Wywołaj metodę `printCar()` na obu obiektach. Uruchom program.
4. Dodaj pole `int maxSpeed`. Dodaj drugi argument do konstruktora ustawiający to pole. Zmodyfikuj metodę `printCar()` aby wyświetlała również maksymalną prędkość.
5. Utwórz dwa obiekty Car używając konstruktora dwuargumentowego.
6. Wywołaj metodę `printCar()` na obu obiektach. Uruchom program.

Autor: Marek Sontag

PAKIET, MODYFIKATORY DOSTĘPU, IMPORT

PAKIET (PACKAGE) – inaczej folder w którym znajduje się klasa



Słowo kluczowe package

Ścieżka odpowiadająca folderom

```
package pl.sda.cars;


class Car {
}
```

MODYFIKATORY DOSTĘPU

Określają co może używać pola lub metody

Autor: Marek Sontag

MODYFIKATORY DOSTĘPU

- `public` – dostęp dla wszystkich
- `protected` – dostęp dla klas dziedziczących oraz w obrębie pakietu
- `private` – dostęp tylko w tej samej klasie
- *package-scope* – dostęp w obrębie pakietu (domyślny)

Autor: Marek Sontag

MODYFIKATORY DOSTĘPU - przykład

```
class Car {
      public String model;
      private String fuel;
}
```

```
public static void main(String[] args) {
      Car myCar = new Car();
      System.out.println(myCar.model);
      System.out.println(myCar.fuel);
}
```

Błąd!

Brak dostępu!

Autor: Marek Sontag

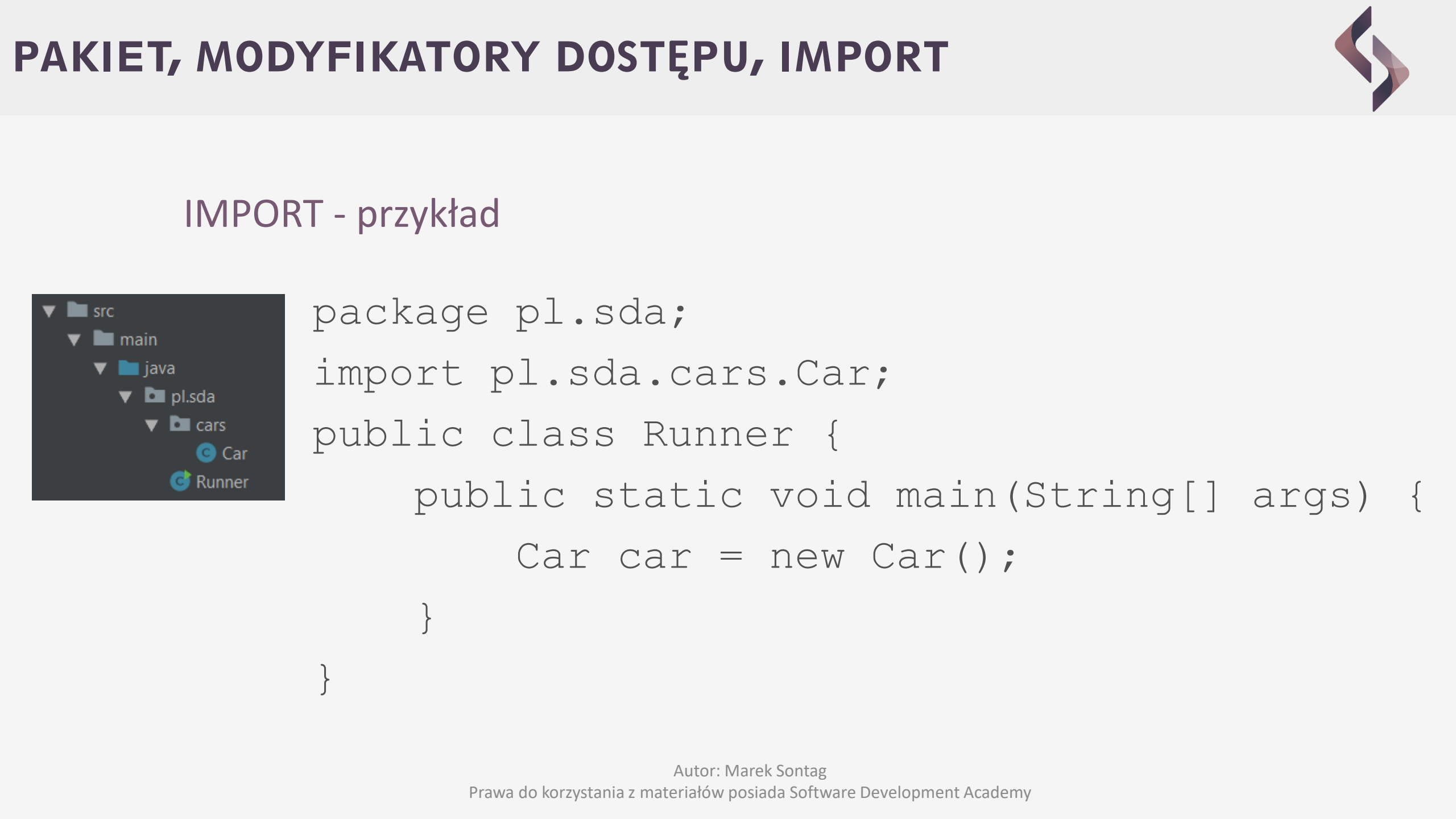
PAKIET, MODYFIKATORY DOSTĘPU, IMPORT

IMPORT

- Deklaracja gdzie znajduje się używana klasa

Autor: Marek Sontag

PAKIET, MODYFIKATORY DOSTĘPU, IMPORT



IMPORT - przykład


```
package pl.sda;
import pl.sda.cars.Car;
public class Runner {
    public static void main(String[] args) {
        Car car = new Car();
    }
}
```

Autor: Marek Sontag

DZIEDZICZENIE

DZIEDZICZENIE – dziecko dziedziczy cechy rodzica

- Klasy mogą dziedziczyć
  - Klasa dziedzicząca przejmuje cechy klasy rodzica
- Klasy dziedziczą pola i metody

Słowo kluczowe

```
class Dziecko extends Rodzic { }
```

Klasa dziedzicząca Klasa rodzica

Autor: Marek Sontag

DZIEDZICZENIE

```
class Car {
	public String name;
	protected int power;
	private String engineType;
}
```

```
class Jeep extends Car {
}
```

```
public static void main(String[] args) {
	Jeep wrangler = new Jeep();
	System.out.println(wrangler.name);
	System.out.println(wrangler.power);
	System.out.println(wrangler.engineType);
}
```

OK

OK, o ile jesteśmy w tym samy pakiecie

BŁAD! engineType jest prywatny!

ZADANIA

Wykonaj zadania ze stron 20 i 21 (Java – Wprowadzenie do języka)

Autor: Marek Sontag

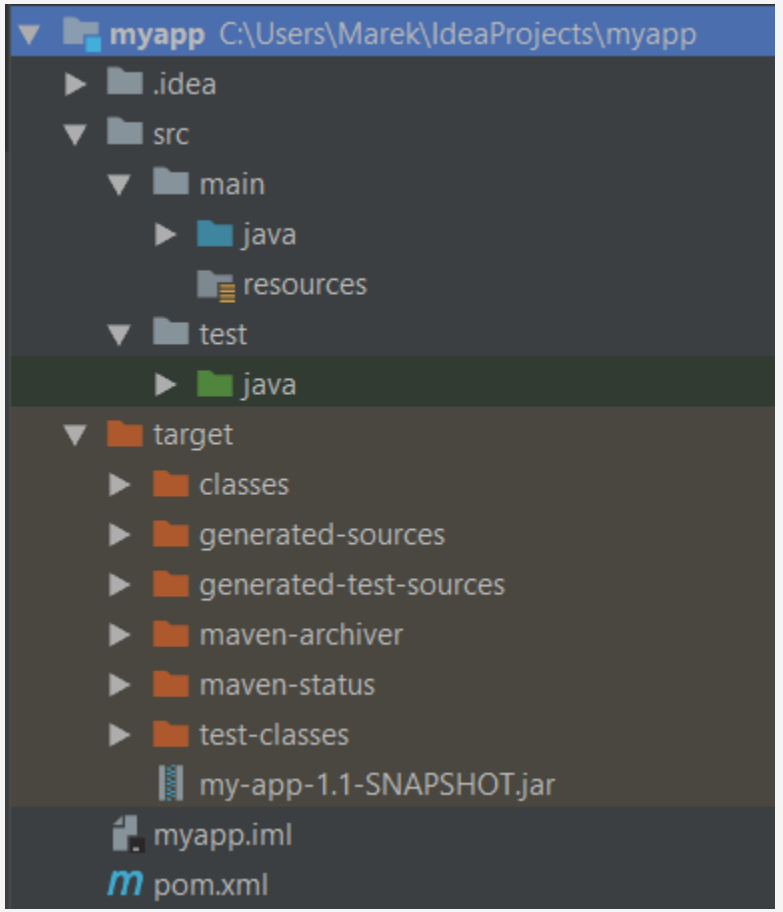
MAVEN – zarządzenie zależnościami

- Programy używają innych programów/bibliotek
- Zależne programy/biblioteki się zmieniają
- Dostępne narzędzia rozwiązujące ten problem:
  - **Maven**
  - Gradle
  - Ant
  - Inne…

Autor: Marek Sontag

MAVEN – KONWENCJA PONAD KONFIGURACJĘ

Ustalona z góry struktura projektu zamiast szczegółowej konfiguracji



Autor: Marek Sontag

MAVEN

POM.XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>pl.sda</groupId>
    <artifactId>my-app</artifactId>
    <version>1.1-SNAPSHOT</version>

</project>
```

MAVEN – ZALEŻNOŚCI

```
<dependencies>
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-lang3</artifactId>
        <version>3.8.1</version>
    </dependency>
</dependencies>
```

MAVEN – BUDOWANIE ARTEFAKTU (default)

1. Validate
2. Compile
3. Test
4. Package
5. Integration-test
6. Verify
7. Install
8. Deploy

Autor: Marek Sontag

MAVEN – CZYSZCZENIE PROJEKTU (clean)

Usuwa zawartość folderu target

Autor: Marek Sontag

ZADANIA

- Utworzyć nowy projekt Maven
- Nadać odpowiednie parametry GAV
- Dodać nową zależność do biblioteki commons-lang3 (sprawdź Maven Central)
- Sprawdzić zawartość folderu .m2 (lokalizacja: folder_użytkownika/.m2)
- W folderze zgodnym z konwencją utworzyć klasę Runner z psvm
- Zainstaluj aplikację w lokalnym repozytorium
- Wykonaj dowolną modyfikację w klasie Runner
- Zwiększ numer wersji w pom.xml
- Usuń zawartość folderu target (clean)
- Ponownie zainstaluj aplikację w lokalnym repozytorium
- Sprawdzić zawartość folderu .m2

Autor: Marek Sontag

CZYM SĄ TESTY?

- Klasy sprawdzające poprawność działania innych klas

Autor: Marek Sontag

PO CO SĄ TESTY?

- Udowadniają poprawność działania programu
- Informują o błędach, regresjach
- Dokumentują kod

Autor: Marek Sontag

BIBLIOTEKI DO TESTOWANIA

- JUnit
- AssertJ
- inne

JAK WYGLĄDA TEST?

- Metoda w klasie testowej z anotacją `@Test`

```
public class TestClass {

    @Test
    public void testMethod() {
    }
}
```

TESTOWANIE

PRZYKŁAD

```
public class CarTest {

    @Test
    public void testModel() {
        Car car = new Car(„Lancer EVO”);

        assertEquals(„Lancer EVO”, car.model);
    }
}
```

STRUKTURA TESTU

- Given
- When
- Then

Autor: Marek Sontag

TESTOWANIE

PRZYKŁAD

```
public class CarTest {

    @Test
    public void testModel() {
        // given
        Car car = new Car();
        // when
        car.setModel(„Mini”);
        // then
        assertEquals(„Mini”, car.getModel());
    }
}
```

ASERCJE

- JUnit
  - `assertEquals(expected, actual)`
  - `assertTrue(warunek-logiczny)`
- AssertJ
  - Fluent:

```
assertThat(frodo.getName()).isEqualTo("Frodo");
assertThat(frodo).isNotEqualTo(sauron);
```

CYKL ŻYCIA

- `@BeforeAll`
- `@BeforeEach`
- `@Test`
- `@AfterEach`
- `@AfterAll`

CO PISZEMY NAJPIERW:
KOD PROGRAMU CZY TEST?

Autor: Marek Sontag

ZADANIA

Wykonaj zadania ze stron 25 i 26 (Java – Wprowadzenie do języka)

Autor: Marek Sontag

DANE W PROGRAMACH - TYPY

TYP OKREŚLA JAKI RODZAJ DANYCH
PRZECHOWUJE KONKRETNA ZMIENNA LUB POLE

```
int liczba = 100;
```


Zmienna `liczba` jest typu
`int` (liczba całkowita)

DANE W PROGRAMACH - TYPY

PODZIAŁ TYPÓW

- Proste (liczbowe, wartość logiczna, znak)
- Złożone

Autor: Marek Sontag

DANE W PROGRAMACH - TYPY

PROSTE LICZBOWE

Stałoprzecinkowe (np. 1, 2019, -35)
- `byte`
- `short`
- `int`
- `long`

Zmiennoprzecinkowe (np. 0.25, -273.15)
- `float`
- `double`

DANE W PROGRAMACH - TYPY

PROSTE LICZBOWE

od -128 do 127 → `byte`

od -32 768 do 32 767 → `short`

od -2^31 do 2^31-1 → `int`

od -2^63 do 2^63-1 → `long`

`float` ← ok. 6-7 cyfr po przecinku

`double` ← ok. 15 cyfr po przecinku

Autor: Marek Sontag

DANE W PROGRAMACH - TYPY

LICZBY ZMIENNOPRZECINKOWE – uwaga na zaokrąglenia!

25 luty 1991 – atak na bazę lotniczą w Dhahranie

- Błędne zachowanie systemu PATRIOT
- Zginęło 28 żołnierzy, ok. 100 rannych

DANE W PROGRAMACH - TYPY

LOGICZNE	`boolean`	← Wartość: `true` lub `false`
ZNAKOWE	`char`	← Znak UNICODE

Autor: Marek Sontag

ZADANIA

Wykonaj zadania ze stron 27-29 (Java – Wprowadzenie do języka)

Autor: Marek Sontag

DANE W PROGRAMACH - TYPY

TYPY ZŁOŻONE (OBIEKTOWE)

- Klasy złożone z innych typów (prostych lub złożonych)

```
class Point {
    int x;
    int y;
}
```

```
class Line {
    Point a;
    Point b;
}
```

Typ złożony

Autor: Marek Sontag

ZADANIA

Wykonaj zadania dla typów złożonych ze strony 30

(Java – Wprowadzenie do języka)

Autor: Marek Sontag

AUTOBOXING - AUTOUNBOXING

TYPY PROSTE I ICH ODPOWIEDNIKI

byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Char

Autor: Marek Sontag

AUTOBOXING - AUTOUNBOXING

AUTOBOXING

```
Integer i = 1;
```


zamiast

```
new Integer(1)
```

AUTOBOXING - AUTOUNBOXING

AUTOUNBOXING

```
Integer i = new Integer(1);

int unboxed = i;
```


zamiast

```
i.intValue()
```

Autor: Marek Sontag

AUTOBOXING - AUTOUNBOXING

AUTOUNBOXING

```
Integer i = new Integer(1);

int unboxed = i;
```


zamiast

```
i.intValue()
```

Autor: Marek Sontag

ZADANIA

- Wykorzystaj projekt z poprzedniego zadania
- Utwórz klasę `Autoboxing`
- W klasie `Autoboxing` zadeklaruj pole `autoboxingExample` typu `Integer` z wartością 1
- Utwórz klasę `AutoboxingTest`
- W teście sprawdź wartość tego pola
- Utwórz klasę `Autounboxing`
- W klasie `Autounboxing` zadeklaruj pole `autounboxingExample` typu `int` z wartością `new Integer(12)`
- Utwórz klasę `AutounboxingTest`
- W teście sprawdź wartość tego pola

PRZETWARZANIE DANYCH - OPERATORY

JAK PRZETWARZAĆ DANE?

Zmienne i dane bez możliwości ich przetwarzania są mało użyteczne

- OPERATORY pozwalają pracować na danych

Autor: Marek Sontag

PRZETWARZANIE DANYCH - OPERATORY

PRZYPISANIE (=)

```
char znak = 'a';
```
← znak ma wartość 'a'

PRZETWARZANIE DANYCH - OPERATORY

ARYTMETYCZNE (+ - * / %)

```
int a = 5 + 5;
int c = a - 2;
int d = 5 * 3;
int e = 10 / 3;
int f = 10 % 3;
```

← dzielenie całkowite (bez reszty: e = 3)

← dzielenie modulo (reszta z dzielenia: f = 1)

Autor: Marek Sontag

PRZETWARZANIE DANYCH - OPERATORY

ARYTMETYCZNE – wersja skrócona

```
int x = 10;
x += 2;
x -= 3;
x *= 4;
x /= 5;
x %= 6;
```

Inaczej: x = x + 2, czyli x = 10 + 2 = 12

Inaczej: x = x - 3, czyli x = 12 – 3 = 9

Inaczej: x = x * 4, czyli x = 9 * 4 = 36

Inaczej: x = x / 5, czyli x = 36 / 5 = 7

Inaczej: x = x % 6, czyli x = 7 % 6 = 1

Autor: Marek Sontag

PRZETWARZANIE DANYCH - OPERATORY

PORÓWNANIA (równy ==)

```
int a = 5;
int b = 5;
boolean x = a == b;
boolean y = a == 6;
```

x = true (a jest równe b)

y = false (a nie jest równe 6)

Autor: Marek Sontag

PRZETWARZANIE DANYCH - OPERATORY

PORÓWNANIA (różny !=)

```
int a = 5;
int b = 10;
boolean x = a != b;
boolean y = a != 5;
```

x = true (a jest różne od b)

y = false (a nie jest różne od 5)

Autor: Marek Sontag

PRZETWARZANIE DANYCH - OPERATORY

PORÓWNANIA (większy >, mniejszy <)

```
int a = 5;
int b = 10;
boolean x = a < b;
boolean y = a > b;
```

x = true (a jest mniejsze od b)

y = false (a nie jest większe od b)

Autor: Marek Sontag

PRZETWARZANIE DANYCH - OPERATORY

PORÓWNANIA (większy lub równy >=, mniejszy lub równy <=)

```
int a = 5;
int b = 10;
boolean x = a <= b;
boolean y = a >= 5;
```

x = true (a jest „mniejsze lub równe” b)

y = true (a jest „większe lub równe” 5)

Autor: Marek Sontag

PRZETWARZANIE DANYCH - OPERATORY

LOGICZNE (AND &&, OR ||, NEGACJA !)

```
boolean prawda = true;
boolean falsz = false;
boolean x = prawda && falsz;
boolean y = prawda || falsz;
boolean z = !falsz;
```

x = false

y = true

z = true

Autor: Marek Sontag

ZADANIA

1. Utwórz nowy projekt (maven). Dodaj dependencje JUnit oraz AssertJ.
2. Utwórz klasę `Calculator`.
3. Utwórz klasę testową `CalculatorTest`.
4. Napisz test metody `int add(int a, int b)` z klasy `Calculator`. Dodaj odpowiednią asercję. Test się nie skompiluje, bo nie ma jeszcze tej metody!
5. Dodaj pustą metodę `int add(int a, int b).` Uruchom test. Test powinien zakończyć się niepowodzeniem.
6. Dopisz ciało metody `add`, tak by wynik był poprawny. Uruchom test.
7. Postępuj podobnie dla innych metod kalkulatora.

Autor: Marek Sontag