Politechnika Wrocławska

# Platformy Programistyczne .NET i Java

## Wprowadzenie do języka JAVA

Dr inż. Radosław Idzikowski

HR EXCELLENCE IN RESEARCH

# Wprowadzenie do Javy

- **Historia i ewolucja** Javy
  - Stworzona przez Jamesa Goslinga w firmie Sun Microsystems w 1995 roku.
  - od 2010 rozwijana przez Oracle Corporation.
- **Filozofia języka**
  - Napisz raz, uruchom wszędzie (Write Once, Run Anywhere - WORA).
  - Bezpieczeństwo i solidność.
  - Wysoka wydajność dzięki JVM.

# Dlaczego Java?

- **Przenośność**
  - Aplikacje działają na różnych platformach bez modyfikacji kodu.
- **Wydajność**
  - JVM z Just-In-Time (JIT) kompilacją.
- **Bezpieczeństwo**
  - Silny model bezpieczeństwa.
- **Bogata biblioteka standardowa**
  - Szeroki zakres gotowych do użycia klas i funkcji.
- **Wsparcie społeczności**
  - Duża liczba programistów i zasobów online.

# Zastosowania Javy

- **Aplikacje desktopowe**
  - Narzędzia i programy użytkowe.
- **Aplikacje serwerowe**
  - Serwery aplikacji, backendy usług webowych.
- **Aplikacje mobilne**
  - Android - główny język programowania.
- **Aplikacje webowe**
  - JSP, Servlets, Spring Framework.
- **Systemy wbudowane**
  - Urządzenia IoT, telewizory, dekodery.

# Narzędzia Java

- JDK (Java Development Kit)
- IDE (Integrated Development Environment)
- Maven i Gradle
- JUnit
- Javadoc

# JDK (Java Development Kit)

- JDK:
  - Podstawowy zestaw narzędzi do tworzenia oprogramowania w języku Java.
  - Zawiera kompilator javac, który przekształca kod źródłowy w kod bajtowy.
  - Zawiera narzędzie java do uruchamiania programów Java.
- JRE (Java Runtime Environment):
  - Środowisko wykonawcze Javy, zawiera JVM i biblioteki standardowe.
  - Wchodzi w skład JDK.
- JVM (Java Virtual Machine):
  - Maszyna wirtualna, która uruchamia kod bajtowy Javy na różnych platformach.

# IDE (Integrated Development Environment)

- `IntelliJ IDEA`:
  - Popularne, bogate w funkcje IDE do Javy.
  - Wersje Community (darmowa) i Ultimate (płatna).
- `Eclipse`:
  - Open-source'owe `IDE`.
  - Silne wsparcie dla pluginów, umożliwiające rozwijanie aplikacji w różnych językach.
- `NetBeans`:
  - `IDE` sponsorowane przez Apache.
  - Zintegrowane wsparcie dla `Javy` i innych języków.
- `VS Code`:
  - Lekki edytor kodu z wieloma wtyczkami dla `Javy`.

# Maven vs Gradle

`Maven` i `Gradle` to najpopularniejsze narzędzia do automatyzacji budowania aplikacji w `Javie`. Oba te narzędzia są często porównywane w celu znalezienia najlepszego rozwiązania. Automatyzacja budowania obejmuje:

- Pobieranie zależności,
- Kompilowanie kodu,
- Pakowanie kodu binarnego,
- Uruchamianie testów,
- Wdrażanie do systemów produkcyjnych.

Procesy te są zautomatyzowane, co oznacza, że mogą być powtarzane bez ingerencji człowieka.

# Maven

Projekt Maven jest prowadzony przez Apache Software Foundation. zajmuje się dwoma aspektami tworzenia oprogramowania – sposobem, w jaki oprogramowanie jest tworzone oraz jego zależnościami. Plik XML opisuje budowany projekt aplikacji, zawiera on:

- zależności do innych zewnętrznych modułów i komponentów,
- kolejność budowy,
- katalogi i wymagane wtyczki,
- zdefiniowane cele do wykonania zdefiniowanych zadań, takich jak kompilacja kodu i jego pakowanie (packaging).

Maven dynamicznie pobiera biblioteki Java i wtyczki z jednego lub więcej repozytoriów i przechowuje je w lokalnej pamięci podręcznej.

# Maven

```
<?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 https://maven.apache.org/xsd/maven-4.0.0.x
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.3.0.RELEASE</version>
    </parent>
    <groupId>pl.bykowski</groupId>
    <artifactId>springboot-buildpacks</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>example-maven</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
```

# JUnit

- `JUnit`:
  - Framework do testowania jednostkowego dla `Javy`.
  - Pozwala na pisanie i uruchamianie testów, zapewniając weryfikację poprawności kodu.
  - Obsługuje adnotacje takie jak `@Test`, `@Before`, `@After`.

# Gradle

Gradle – to system open source, który automatyzujący kompilację oprogramowania oparty na koncepcjach Apache Ant i Apache Maven. Ponadto wykorzystuje on język dziedzinowy (domain-specific language DSL – język przystosowany do rozwiązywania określonej dziedziny problemów ) oparty na Groovy.
Zamiast formy XML używanej przez Apache Maven do deklarowania konfiguracji projektu używa dedykowanego pliku .gradle:

# Gradle

```
plugins {
  id 'org.springframework.boot' version '2.3.0.RELEASE'
  id 'io.spring.dependency-management' version '1.0.9.RELEASE'
  id 'java'
}
group = 'pl.bykowski'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '1.8'
repositories {
  mavenCentral()
}
dependencies {
  implementation 'org.springframework.boot:spring-boot-starter-web'
  testImplementation('org.springframework.boot:spring-boot-starter-test') {
    exclude group: 'org.junit.vintage', module: 'junit-vintage-engine'
  }
}
test {
  useJUnitPlatform()
}
```

Witać tu analogie, gdzie również wskazujemy na wersje Java, informacje na temat zależności. Warto pamiętać, że `gradle` pobiera informacje na temat zależności z repozytorium `Maven` co widać przy `repositories`.
Aby uniknąć niejasności – `Maven` i `Maven Repository` to zupełnie dwa oddzielne byty. Pierwszy z nich to system budowy, natomiast drugi to globalne repozytorium (magazyn) gdzie składowane są artefakty (zależności), które później możemy uwzględnić w projekcie.

# Javadoc

- Javadoc:
  - Narzędzie do generowania dokumentacji API bezpośrednio z kodu źródłowego.
  - Komentarze dokumentacyjne w kodzie są przekształcane w czytelną dokumentację HTML.
  - Ważne adnotacje: `@param`, `@return`, `@see`.

# Składnia języka Java

```
class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello World!");
    }
}
```

```
$ javac HelloWorld.java
$ ls
HelloWorld.class HelloWorld.java
$ java HelloWorld
Hello World!
```

# Składnia języka Java

```
public class Hello
{
    public static void main(String[] args)
    {
        System.out.println("Hello World!");
    }
}
```

```
$ javac HelloWorld.java
HelloWorld.java:1: error: class Hello is public, should be declared in a file named Hello.java
public class Hello
       ^
1 error
```

# Składnia języka Java

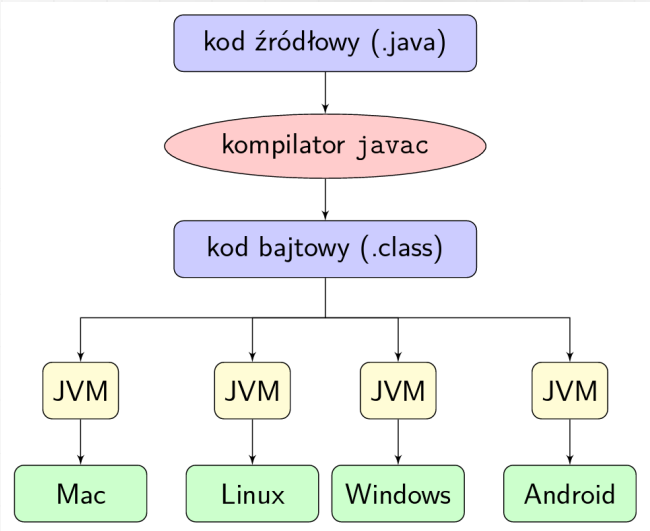
```
class Foo
{
    public static void main(String[] args)
    {
        System.out.println("Hello␣from␣Foo␣class");
    }
}

class Bar
{
    public static void main(String[] args)
    {
        System.out.println("Hello␣from␣Bar␣class");
    }
}
```

```
$ javac Hello.java
$ ls *.class
Foo.class Bar.class
$ java Foo
Hello from Foo class
$ java Bar
Hello from Bar class
```

# Składnia języka Java

- W pliku `.java` może być wiele klas, ale tylko jedna publiczna.
- Dla wywołania `javac HelloWorld.java` kompilator będzie poszukiwał w pliku publicznej klasy o nazwie `HelloWorld`

# Kompilacja oraz uruchomienie



# Typy danych

Java to język o ścisłej kontroli typów– każda zmienna musi mieć określony typ. W Javie istnieje osiem podstawowych typów:

- typy całkowite
  - `int`
  - `short`
  - `long`
  - `byte`
- typy zmiennoprzecinkowe
  - `float`
  - `double`
- `char`
- `boolean`

# Słowo kluczowe `var`

- Wprowadzone w `Java 10`.
- Wspiera nową cechę języka `local variable type inference`.
- Kompilator `Java` rozpoznaje typ zmiennej w oparciu o kontekst wykorzystania.
- Programista nie musi deklarować typu wprost.
- Możliwość wykorzystania tylko dla lokalnych zmiennych w funkcjach.
- Zmienna typu `var` musi być zawsze inicjalizowana wartością.

# Typ `String`

`String` w języku `Java` jest specjalną klasą do przechowywania niemodyfikowalnych ciągów znaków. W języku `Java` obiekt typu `String` może zostać utworzony jako:

- literał– obiekt (istniejący) w tzw. `String pool`,
- obiekt (zawsze nowy) tworzony przy pomocy operatora `new` na stercie.

# Typ String

Klasa String zawiera m.in. metody:

- `int length()` -– zwraca długość łańcucha,
- `boolean equals(Object other)` -– zwraca `true`, jeżeli łańcuch jest identyczny z łańcuchem `other`,
- `int compareTo(Object other)` -– porównuje dwa łańcuchy w sensie leksograficznym,
- `boolean startsWith(String prefix)` -– zwraca wartość `true`, jeżeli na początku łańcucha znajduje się przyrostek `prefix`,
- `boolean endsWith(String suffix)` -– zwraca wartość `true`, jeżeli na końcu łańcucha znajduje się przyrostek suffix,
- `String toLowerCase()` -– zwraca nowy łańcuch z literami przekonwerowanymi na małe,
- `String toUpperCase()` -– zwraca nowy łańcuch z literami przekonwerowanymi na wielkie,
- `String trim()` -– zwraca nowy łańcuch z usuniętymi białymi znakami na początku i końcu łańcucha.

# Typ String

Do efektywnej konkatenacji ciągów znaków (operator + dla typu `String`) należy wykorzystywać klasę `StringBuilder`.

```
class TestString
{
    public static void main(String[] args)
    {
        StringBuilder sb = new StringBuilder();
        sb.append("foo, ");
        sb.append("bar, ");
        sb.append("baz");
        String s = sb.toString() + "!!!";
        System.out.println(s);
    }
}
```

# Operatory

- Operator przypisania,
- Operatory arytmetyczne,
- Operatory inkrementacji i dekrementacji,
- Operatory relacyjne i logiczne,
- Operatory bitowe,
- Operator rzutowania.

# Definiowanie klas

Prosta klasa `Project`, która może stanowić podstawę systemu obsługi projektów:

```
class Project
{
    private String title;

    public Project(String t)
    {
        title = t;
    }
    public String getTitle()
    {
        return title;
    }
}
```

# Konstruowanie obiektów

Konstruktory definiują początkowy stan obiektów. Konstrukcję obiektów wspiera kilka różnych mechanizmów:

- przeciążenie,
- domyślna inicjalizacja pól,
- konstruktor bezargumentowy,
- jawna inicjalizacja pól,
- wywołanie innego konstruktora.

# Konstruowanie obiektów

Cechy konstruktorów w `Javie`:

- nazwa konstruktora musi być taka sama jak nazwa klasy,
- klasa może mieć więcej niż jeden konstruktor,
- konstruktor może przyjmować zero lub więcej parametrów,
- konstruktor nie zwraca wartości,
- wywołanie konstruktora odbywa się zawsze przy użyciu operatora `new`.

# Pola i metody statyczne

- Pola statyczne -– należy do klasy, a nie do konkretnego obiektu.
- Stałe statyczne

```
public class Math
{
    public static final double PI = 3.14...;
    ...
}
```

- Metody statyczne mają dwojakie zastosowanie:
  - kiedy metoda potrzebuje dostępu tylko do pól statycznych,
  - kiedy metoda nie wymaga dostępu do stanu obiektu (np. `Math.pow`).

# Klasa generyczna ArrayList

- Deklaracja i utworzenie listy tablicowej przechowującej obiekty typu `Project`

```
ArrayList<Projects> projs = new ArrayList<>();
```

- Nowe elementy można dodać do listy za pomocą metody `add`

```
projs.add(new Project("PL-Grid"));
projs.add(new Project("SpinLab"));
```

- Metoda `size` pozwala na określenie rozmiaru listy

```
projs.size();
```

# Klasa generyczna ArrayList

Dostęp do elementów listy jest realizowany przez metody get i set.

- Aby ustawić wartość i-teg elementu należy:

```
projs.set(i, p1);
```

- Aby pobrać wartość elementu listy należy:

```
Project p = projs.get(i);
```

# Pakiety

Klasy w Javie można segregować w tzw. pakiety przez oddzielenie klas od pozostałych. Umieszczenie klasy w pakiecie polega na dodaniu nazwy wybranego pakietu na początku pliku źródłowego:

```
package apps.projects;
public class Project
{
    // ...
}
```

# Pakiety

Dostęp do pakietu z innego kodu źródłowego można uzyskać poprzez użycie instrukcji `import`:

```
import apps.projects.Task;
import apps.projects.Project;

class ProjectManger
{
    // ...
}
```

# Dziedziczenie

- Dziedziczenie to technika umożliwiająca tworzenie nowych klas na bazie już istniejących.
- Klasa dziedzicząca po innej klasie przejmuje jej metody i pola oraz dodaje własne metody i pola.
- Relacja dziedziczenia jest wyrażana przez słowo kluczowe `extends`:

```
public class Project extends Base
{
    // ...
}
```

- W języku Java dziedziczenie jest wyłącznie publiczne.
- Java nie obsługuje dziedziczenia wielokrotnego.

# Stos i sterta

- Stos:
  - obszar pamięci wykorzystywany dla wykonywania wątków,
  - zawiera zmienne typów prostych oraz referencje do obiektów zaalokowanych na stercie.
- Sterta:
  - obszar pamięci wykorzystywany do alokacji pamięci dla obiektów (`new`),
  - dzieli się na dwa obszary 0– `Young Generation` i `Old Generation`,
  - `Garbage Collection` – zwalnianie pamięci dla obiektów, które nie posiadają żadnej referencji.

# Sterta -– Young Generation

- Young Generation jest miejscem, w którym tworzone są nowe obiekty.
- Gdy obszar Young Generation się zapełni wykonywane jest garbage collection (Minor GC).
- Young Generation dzieli się na trzy części -– obszar Eden Memory oraz dwa obszary Survivor Memory.

# Sterta -– Young Generation

- większość nowo tworzonych obiektów trafia do przestrzeni `Eden Memory`,
- gdy przestrzeń `Eden Memory` jest wypełniona obiektami uruchamiany jest `Minor GC` i wszystkie obiekty, które przetrwały zostaną przeniesione do jednego z obszarów `Survivor Memory`,
- `Minor GC` sprawdza także obiekty, które przetrwały i przenosi je do innego obszaru `Survivor Memory`, więc jeden z obszarów `Survivor Memory` jest zawsze pusty,
- Obiekty, które przetrwały wiele cykli Minor GC zostają przeniesione do obszaru `Old Generation`.

# Sterta -– Old Generation

- Obszar pamięci Old Generation zawiera obiekty o długim czasie życia, które przetrwały wiele cykli Minor GC.
- Zwykle garbage collection jest wykonywane na obszarze Old Generation, gdy się zapełni.
- Garbage collection na obszarze Old Generation jest nazywane Major GC i zajmuje zwykle dłuższy czas.
- Wszystkie operacje garbage collection są zdarzeniami Stop The World – wszystkie wątki aplikacji są zatrzymywane do czasu zakończenia operacji.

# Stos i sterta

```
public class StackHeap
{
    public static void main(String[] args)
    {
        Thing t = new Thing();
        int a = 2;
        updateThing(t);
    }
    private static void updateThing(Thing t)
    {
        t.incr();
    }
}
```

# Stos i sterta

- Dla wywołania funkcji `main` alokowany jest obszar pamięci na stosie, który będzie wykorzystywany przez funkcję `main`.
- Dla obiektu typu `Thing` alokowana jest pamięć na ten obiekt na stercie oraz alokowana jest pamięć na stosie dla referencji do tego obiektu.
- Dla obiektu typu podstawowego `int` alokowana jest pamięć na stosie.
- Dla wywołania funkcji `updateThing` alokowany jest obszar pamięci na stosie, który będzie wykorzystywany przez funkcję `updateThing`.

# Klonowanie obiektów

- Po skopiowaniu zmiennej przechowującej referencje do obiektu oryginał i kopia są referencjami do tego samego obiektu.
- Zmiana jednej z nich pociąga za sobą zmianę w drugiej.

```
Thing t1 = new Thing();
t1.decr();
System.out.println("Thing t1: " + t1);
Thing t2 = t1;
t2.decr();
System.out.println("Thing t1: " + t1);
System.out.println("Thing t2 (t1 copy): " + t2);
```

# Klonowanie obiektów

Aby utworzona kopia była całkiem nowym obiektem należy użyć metody `clone`. W tym celu klasa musi:

- implementować interfejs `Cloneable`,
- przedefiniować metodę clone z modyfikatorem `public`.

```
class Thing implements Cloneable
{
    // ...
    @Override
    public Object clone() throws CloneNotSupportedException
    {
        return super.clone();
    }
}
```