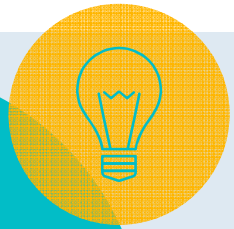
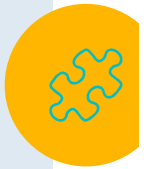
Budowanie oprogramowania – Gradle

dr inż. Marcin Szeląg
Instytut Informatyki, PP

Gradle – wprowadzenie

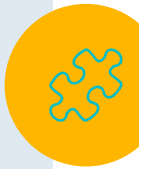


- System **automatyzacji procesu budowania oprogramowania** oraz język opisu tego procesu (*Gradle build language*)
- Powstał w **2007 r.**
- Wymaga Java **JDK** , w wersji **8** lub wyższej
- Obsługuje budowanie **prostych projektów** oraz **projektów złożonych** z podprojektów (*multiproject builds*)
- Działa **przyrostowo** (incremental builds) => szybkość budowania

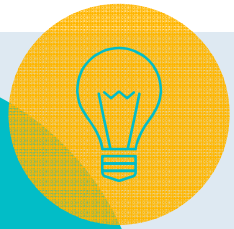


Gradle – wprowadzenie

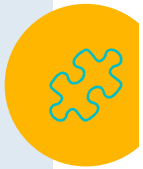
- Obsługuje **pluginy**; wiele gotowych pluginów + możliwość definiowania własnych
- Tworzy **acykliczny graf skierowany** (DAG) zadań (tasków) do wykonania w ramach build'a
- Zarządza **zależnościami przechodnimi**
- Współpracuje z **repozytoriami Apache Maven** (starszy system budowania, powstały w 2004 r.)



Gradle – wprowadzenie

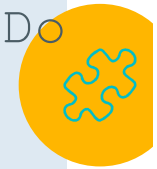


- Wspiera skrypty napisane w **Apache Ant** (jeszcze starszy system budowania, powstały w 2000 r.)
- **Integracja** z Eclipse, IntelliJ IDEA, Android Studio (oficjalny system budowania!) , **…**
- **Integracja** z serwerami ciągłej integracji (np. z Jenkins'em)
- **„samowystarczalność"** dzięki technice **Gradle wrapper** (*self-provisioning build env.*)
- Bogaty **interfejs command-line**



- **Skany budowania** (*build scans*) – publiczne w wersji darmowej, prywatne w wersji Gradle Enterprise

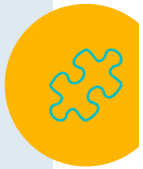
```
BUILD SUCCESSFUL in 1m 20s

7 actionable tasks: 7 executed

Publishing a build scan to
scans.gradle.com requires accepting the
Gradle Terms of Service defined at
https://gradle.com/terms-of-service. Do
you accept these terms? [yes, no] yes

Gradle Terms of Service accepted.

Publishing build scan...

https://gradle.com/s/m4duhyads6erm
```



Gradle – wprowadzenie

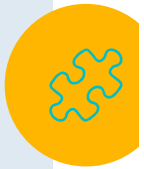
- **Cache procesu budowania** (*build cache*) – domyślnie lokalny (katalog `.gradle`); współdzielony „zdalny” cache w wersji Gradle Enterprise
- **Skrypty** opisujące proces budowania pisane w języku Apache Groovy lub Kotlin (od Gradle 5.0)

- W dalszej części prezentacji omówione zostaną **wybrane elementy języka Groovy**



Apache Groovy

- Język obiektowy powstały w **2003 r.**
- W wielu miejscach **rozszerza funkcjonalność Javy** i pozwala tworzyć bardziej zwięzły kod
- Kod kompilowany do bytecode'u **maszyny wirtualnej Javy (JVM)**
- http://groovy-lang.org



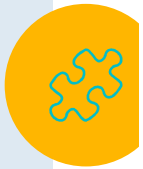
Apache Groovy – cechy

- Każda **wartość podstawowa** (*primitive*) jest **obiektem**
- **Opcjonalność** nawiasów, średników, return, public (http://groovy-lang.org/semantics.html#_optionality)
- Dynamiczne ustalanie wywoływanej metody **podczas wykonania programu** (http://groovy-lang.org/differences.html, pkt. 2. Multi-methods), na podstawie **faktycznego** (a nie deklarowanego) typu obiektu
- **Interpolowane ciągi znaków** (**GStrings**): `"Greetings from the $hello.name task."`
- **Zwykłe ciągi znaków**: `'abc'`

Apache Groovy – cechy

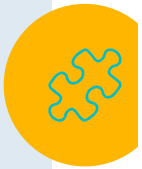
- Klasy i metody bez specyfikatora dostępności są publiczne (***public***)
- **Automatyczne własności** (property) obiektów:

```
class Person {
    String name
}

Person p = new Person()
p.setName("John")
System.out.println(p.getName())
```



Apache Groovy – cechy

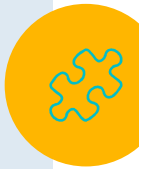
- **Nazwane parametry** (`http://groovy-lang.org/objectorientation.html#_named_parameters_2`) – automatyczne wywołanie metody z pierwszym parametrem typu **`Map`** gdy wykorzystywane są nazwane parametry
- działanie **operatora "=="**: jeżeli a i b implementują `Comparable`, to oblicz `a.compareTo(b)==0`; w przeciwnym razie, oblicz `a.equals(b)`
- sprawdzanie **identyczności** obiektów: `a.is(b)`
- Dodatkowe słowa kluczowe **as**, **def**, in, trait



Apache Groovy – cechy

Listy (http://groovy-lang.org/syntax.html#_lists):

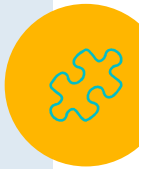
```
def heterogeneous = [1, "a", true];

def letters = ['a', 'b', 'c', 'd']
assert letters[0] == 'a'
assert letters[-1] == 'd'
letters << 'e'
assert letters[4] == 'e'
assert letters[1,3] == ['b', 'd']
assert letters[2..4] == ['c','d','e']
```



Apache Groovy – cechy

- Tablice (http://groovy-lang.org/syntax.html#_arrays):

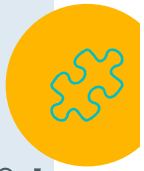
```
String[] arrStr = ['Ananas', 'Banana', 'Kiwi']
assert arrStr instanceof String[]
assert !(arrStr instanceof List)
def numArr = [1, 2, 3] as int[]
assert numArr instanceof int[]
assert numArr.size() == 3
```



Apache Groovy – cechy

- Mapy (http://groovy-lang.org/syntax.html#_maps):

```
def colors = [red: '#FF0000', green:
'#00FF00']
assert colors['red'] == '#FF0000'
assert colors.green == '#00FF00'
colors['pink'] = '#FF00FF'
colors.yellow = '#FFFF00'
assert colors.pink == '#FF00FF'
assert colors['yellow'] == '#FFFF00'
```

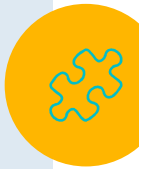


Apache Groovy – cechy

- **Closures** (http://groovy-lang.org/closures.html) – **kluczowy element języka**!
- **Przeciążanie operatorów** (http://groovy-lang.org/operators.html#Operator-Overloading)
- **Groovy Truth** (http://docs.groovy-lang.org/latest/html/documentation/core-semantics.html#Groovy-Truth) - specjalne **reguły dla koercji typów do typu boolean**

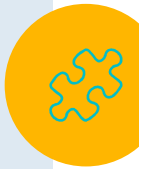
Apache Groovy – cechy

- **Metaprogramowanie**:
  - **Compile-time** metaprogramming (http://groovy-lang.org/metaprogramming.html#_compile_time_metaprogramming) – generowanie kodu na etapie kompilacji
  - **Runtime** metaprogramming (http://groovy-lang.org/metaprogramming.html#_runtime_metaprogramming) – generowanie własności, metod i klas na etapie działania programu
- Dokumentacji API języka Groovy (**Groovy Development Kit**, GDK): http://groovy-lang.org/gdk.html



Apache Groovy – referencje

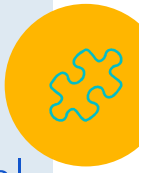
- **Groovy Tutorial** (~1h): https://youtu.be/B98jc8hdu9g
- **Różnice w stosunku do języka Java**: http://groovy-lang.org/differences.html



Apache Groovy – referencje

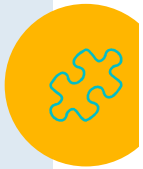
Specyfikacja języka Groovy: http://groovy-lang.org/documentation.html, w szczególności:

- **Syntax**: http://groovy-lang.org/syntax.html
- **Operators**: http://groovy-lang.org/operators.html
- **Program Structure**: http://groovy-lang.org/structure.html
- **Object Orientation**: http://groovy-lang.org/objectorientation.html
- **Closures**: http://groovy-lang.org/closures.html
- **Semantics**: http://groovy-lang.org/semantics.html



Gradle – instalacja

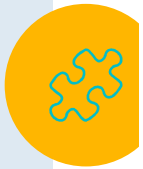
- **Opis ogólny** instalacji: https://docs.gradle.org/current/userguide/installation.html
- **Instalacja manualna** – ustawienie zmiennych:
  - **GRADLE_HOME**
  - **PATH** – dodanie GRADLE_HOME/bin
- **Weryfikacja** instalacji: **gradle -v**



Gradle – tworzenie nowych build'ów*

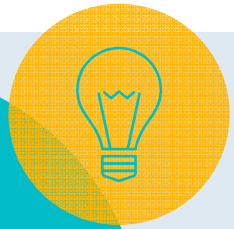
- **`gradle init`** – wygenerowanie standardowych plików Gradle'a (inicjalizacja „pustego" projektu), np.

  `gradle init --type java-library` (lub **`java-application`**)
- domyślny skrypt budujący dla języka Groovy: **`build.gradle`**
- katalog domowy użytkownika Gradle'a: **`USERPROFILE/.gradle`**
- cache projektu: katalog **`.gradle`**

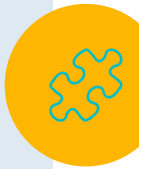


* https://guides.gradle.org/creating-new-gradle-builds

Gradle – tworzenie nowych build'ów*



- Dostępne predefiniowane **zadania** (*tasks*); niektóre działają po podłączenia **pluginów**
- Możliwość definiowania **własnych zadań**
- Podłączanie pluginów – sekcja **`plugins`** na górze skryptu `build.gradle`
- **`base`** plugin (np. definiuje typ zadania **`Zip`**)
- **`gradle tasks`** – komenda pokazująca tylko zadania przypisane do konkretnych grup;
- **`gradle tasks --all`** – pokazuje wszystkie zadania



* https://guides.gradle.org/creating-new-gradle-builds

Gradle – interfejs wiersza poleceń*

- Typowe zadania: **build**, **run**, **clean**
- Składnia (zgodnie z `gradle --help`):

  `gradle [option...] [task...]`

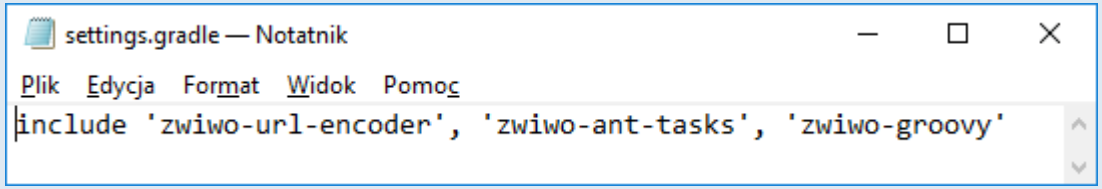
  np.

  `?> gradle --scan build`
- Wyświetlenie uruchomionych procesów Gradle'a (`gradle deamons`): **`gradle -status`**
- Ciągłe budowanie – np. **`gradle --continuous test`** (uwaga – nie uwzględnia zmian w pliku `build.gradle`!)
- Podgląd podprojektów: **`gradle projects`**

* https://docs.gradle.org/current/userguide/command_line_interface.html

Gradle – projekty złożone*

- Struktura projektu złożonego – określona w pliku **`settings.gradle`**, np.:



- Podprojekty mogą być w różnych językach programowania
- Podprojekty mogą mieć swoje skrypty `build.gradle`
- W celu wykonania zadania z podprojektu, należy użyć kwalifikowanej nazwy zadania:

  `[:]subproject-name:task-name`

* https://docs.gradle.org/current/userguide/intro_multi_project_builds.html

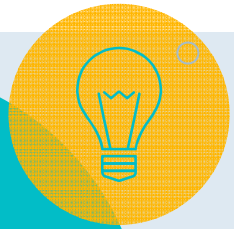
Gradle – Gradle wrapper*

- **Gradle Wrapper** – skrypt wywołujący określoną wersję gradle'a, ew. wcześniej ją pobierając.
- **Struktura projektu** wykorzystującego wrappera:

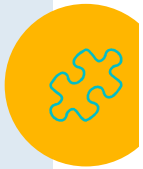
```
├── build.gradle
├── settings.gradle
├── gradle
│   └── wrapper
│       ├── gradle-wrapper.jar
│       └── gradle-wrapper.properties
├── gradlew
└── gradlew.bat
```

* https://docs.gradle.org/current/userguide/gradle_wrapper.html

Gradle – Gradle wrapper*



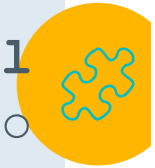
- **`Stworzenie powyższej struktury`** (potrzebna **pełna instalacja** Gradle'a):
  `?> gradle wrapper`
- Z dodatkowymi opcjami:
  `?> gradle wrapper --gradle-version 5.0 --distribution-type all`
- Domyślnie gradle wrapper będzie używał do build'ów tej samej wersji Gradle'a, która została użyta do wygenerowania plików wrapper'a
- Katalog **`gradle`** oraz skrypty **`gradlew`** i **`gradlew.bat`** należy **wgrać do repozytorium**!
- Ustandaryzowane budowanie projektu (**bez instalacji Gradle'a**): **`gradlew build`**

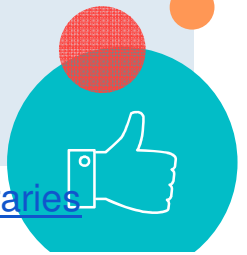


* https://docs.gradle.org/current/userguide/gradle_wrapper.html

Gradle – tworzenie aplikacji i bibliotek Java*

- **`gradle init --type java-application`**
  - zadanie `init` uruchamia zadanie `wrapper`
  - zależności pobierane automatycznie: **`USER_HOME\.gradle\caches\modules-2\files-2.1`**
- **`gradle init --type java-library`**
- **`gradle init --type java-library --dsl kotlin`** (Kotlin DSL zamiast domyślnego Groovy DSL – powstanie plik **`build.gradle.kts`**)

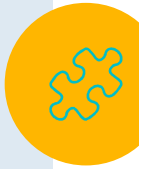




* https://guides.gradle.org/building-java-applications, https://guides.gradle.org/building-java-libraries

Gradle – struktura projektu w Javie


```
├── build
├── gradle
├── src
│   └── main
│       └──java
│       └──resources
│   └── test
│       └──java
│       └──resources
├── build.gradle
├── gradlew
├── gradlew.bat
└── settings.gradle
```



Gradle – podstawowe pluginy*

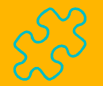
- **Build Init** – definiuje zadania **init**, **wrapper**
- **Base** – dodaje typowe zadania, związane z cyklem życia projektu, np. **build**, **clean**, **check**)
- **Java** – dodaje zadania: **compileJava**, **jar**, **javadoc**, itd. --> diagram zależności; dodaje m.in. konfigurację⁺ **implementation**)
  - **Java Library** (rozszerza Java plugin - dodaje m.in. konfigurację **api**)
- **War** – dodaje zadanie **war**
- **Eclipse** – generuje pliki umożliwiające wczytanie projektu do **Eclipse IDE**
- **IntelliJ IDEA** – generuje pliki umożliwiające wczytanie projektu do **IntelliJ IDEA**

⁺ konfiguracja – określa zakres oddziaływania (*scope*) zależności

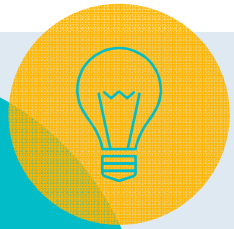
* https://docs.gradle.org/current/userguide/plugin_reference.html

Gradle – domyślne importy w build.gradle

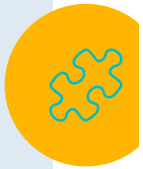
- Lista: https://docs.gradle.org/current/userguide/writing_build_scripts.html#script-default-imports



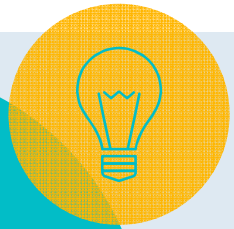
Gradle – przykładowe pliki



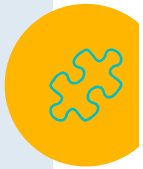
- https://github.com/ruleLearn/rulelearn/blob/master/build.gradle
- https://github.com/ruleLearn/rulelearn/blob/master/settings.gradle
- https://github.com/ruleLearn/rulelearn/blob/master/gradle.properties

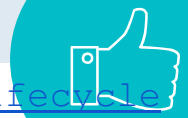


Gradle – cykl budowania projektu



- Budowanie projektu składa się z **3 faz**:
  - **inicjalizacja**
  - **konfiguracja**
  - **wykonanie**

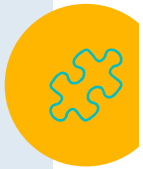




https://docs.gradle.org/current/userguide/build_lifecycle.html#build_lifecycle

Gradle – cykl budowania projektu

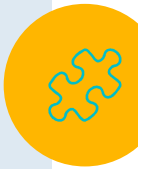
- **Faza inicjalizacji** – Gradle określa jakie projekty składowe biorą udział w budowaniu złożonego projektu (**`settings.gradle`**) i tworzy instancję interfejsu Project dla każdego takiego projektu
- **Faza konfiguracji** – wykonanie skryptów budujących (**`build.gradle`**) wszystkich projektów składowych; konfiguracja wszystkich obiektów typu Project, w tym zadań; zbudowanie pełnego grafu zależności pomiędzy zadaniami (DAG)



Gradle – cykl budowania projektu

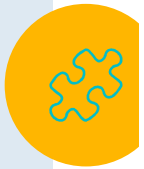
- **Faza wykonania** – określenie podzbioru zadań do wykonania, na podstawie **parametrów wiersza poleceń** (podanych nazw zadań) i **bieżącego katalogu**; wykonanie każdego z tych zadań, w podanym porządku

- **Przykład**: https://docs.gradle.org/current/userguide/build_lifecycle.html#sec:settings_file – Example 1



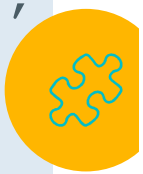
Gradle – cykl budowania projektu

- **`build.gradle`** – własności (properties) + wywołania metod dotyczą obiektu typu `Project` (wykorzystanie mechanizmu delegacji Groovy'ego)
- **`settings.gradle`** – własności (properties) + wywołania metod dotyczą obiektu typu `Settings`; zawsze warto ustawić własność **`rootProject.name`**, nawet dla prostych projektów (bez podprojektów), np.:
- **`rootProject.name = 'ruleLearn'`**



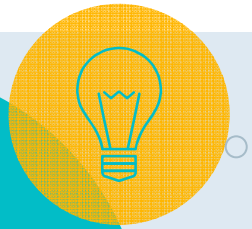
Gradle – budowanie projektu złożonego (multi-project build)*

- **Kompozycja projektu** (drzewo podprojektów względem projektu głównego) definiowana (definiowane) w pliku `settings.gradle`
  - Każdy podprojekt posiada (konfigurowalną) **ścieżkę**
  - **Kompozycja hierarchiczna**: `include 'project1', 'project2:child', 'project3:child:child'` (wystarczy podać liście)
  - **Kompozycja „płaska”**: `includeFlat 'project1', 'project2'`
  - **`gradle projects`** – podgląd struktury projektu złożonego
  - Gradle definiuje **algorytm lokalizowania pliku `settings.gradle`** -> szczegóły



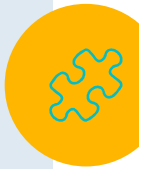
* https://docs.gradle.org/current/userguide/intro_multi_project_builds.html

Gradle – budowanie projektu złożonego



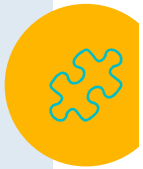
- W pliku `settings.gradle` możliwa jest **modyfikacja elementów drzewa projektów**, np.:

```
rootProject.name = 'main'
project(':projectA').projectDir = new
  File(settingsDir, '../my-project-a')
project(':projectA').buildFileName =
  'projectA.gradle'
```



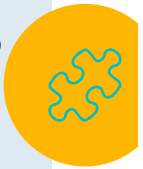
Gradle – własności projektu

- **`gradle properties`** – wyświetlenie własności budowanego projektu

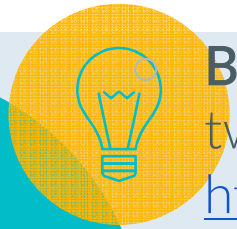


Gradle – integracja z innymi narzędziami

- Dwa sposoby **integracji z Gradle'm**:
  - Gradle używa innego narzędzia – z wykorzystaniem **pluginu do Gradle'a**
  - Narzędzie używa Gradle'a do budowania – z wykorzystaniem **Tooling API** oferowanego przez Gradle'a (np. `Eclipse Buildship plugins`)



Gradle – referencje



- **Breaking Open: Gradle** (wywiad z Hans'em Dockter'em, twórcą Gradle'a): https://www.youtube.com/watch?v=XXoIzzcJr80
- **Cechy Gradle'a**: https://gradle.org/features/
- **Konkretne przykłady – lekcje** (Gradle Tutorials and Guides): https://gradle.org/guides/
- **Gradle User Manual** : https://docs.gradle.org/current/userguide/userguide.html
- **Własności (properties) w języku Groovy:** http://groovy-lang.org/objectorientation.html#properties
- **Gradle API**: https://docs.gradle.org/current/javadoc – najważniejszy interfejs: **`Project`**