Eclipse IDE

The World In and Around

ECLIPSE

Spis treści

1. Wstęp
2. Instalacja Eclipse + Java
3. Pierwsze starcie
4. Pierwszy projekt
5. Konfiguracja uruchomieniowa
6. Dołączanie bibliotek
7. Javadoc
8. Debugger
9. Tworzenie plików JAR
10. Pobieranie wtyczek
11. Podstawowe skróty
12. Porady i wskazówki

Wstęp

- Eclipse jest darmową platformą napisaną w Javie. Dzięki zastosowaniu technologii Java, Eclipse dostępne jest dla wszystkich platform posiadających własną interpretację wirtualnej maszyny Javy, np. Linux, Windows, Solaris, OSx, QNX itp.
- Przy wykorzystaniu odpowiednich wtyczek, Eclipse można zastosować do innych języków dzięki czemu jest elastyczny.
- Specjalna licencja EPL sprawia, że każdy może wpływać na rozwój Eclipse.

- OTI – twórca VA4J (Visual Age for Java) – narzędzia do programowania w Javie napisane w Smalltalk. Można by powiedzieć, że Eclipse to VA4J napisane od nowa w Javie.
- Pierwsza wersja Eclipse 1.0 – listopad 2001
- Na poważnie używana od wersji 3.0
- Projekt stworzony przez firmę IBM, obecnie współpracują nad nim m.in. takie przedsiębiorstwa jak IBM, Oracle czy Intel.
- San IBM zainwestował $40 000 000 w prace nad Eclipse.

Instalacja IDE

- Odwiedzamy stronę projektu **eclipse.org**, po czym przechodzimy do działu Downloads.
- Wyświetlona zostanie strona z listą dostępnych wersji platformy, m.in. :
- - **„Eclipse for Java Developers”,**
- - „Eclipse IDE for Java EE Developers”,
- - „Eclipse IDE for C/C++ Developers”,
- - „Eclipse for Mobile Developers”,
- - etc.

Visit other Eclipse Sites

Eclipse Downloads

Eclipse Juno (4.2) SR1 Packages for Windows

Eclipse IDE for Java EE Developers, 221 MB
Downloaded 254,347 Times Details
Windows 32 Bit
Windows 64 Bit

Eclipse Classic 4.2.1, 183 MB
Downloaded 161,536 Times Details Other Downloads
Windows 32 Bit
Windows 64 Bit

Eclipse IDE for Java Developers, 150 MB
Downloaded 126,452 Times Details
Windows 32 Bit
Windows 64 Bit

Spring Tool Suite Promoted Download
Complete IDE for enterprise Java, Spring, Groovy, Grails and the Cloud.
Download

Eclipse IDE for C/C++ Developers, 129 MB
Downloaded 65,180 Times Details
Windows 32 Bit
Windows 64 Bit

Eclipse for Mobile Developers, 144 MB
Downloaded 41,950 Times Details
Windows 32 Bit
Windows 64 Bit

Eclipse IDE for Java and DSL Developers, 260 MB
Downloaded 38,319 Times Details
Windows 32 Bit
Windows 64 Bit

Eclipse for RCP and RAP Developers, 227 MB
Downloaded 28,151 Times Details
Windows 32 Bit
Windows 64 Bit

Installing Eclipse

- Install Guide
- Compare/Combine Packages
- Known Issues
- Updating Eclipse

Related Links

- Documentation
- Make a Donation
- Forums
- Eclipse Juno (4.2)
- Eclipse Indigo (3.7)

- Wybieramy interesującą nas wersje (Juno, Indigo, **Helios**, Galileo...) oraz system operacyjny, na którym pracujemy.
- Po pobraniu platformy, należy ją wypakować w dowolnym miejscu na dysku.
- Po wypakowaniu program jest gotowy do działania, bez jakiejkolwiek instalacji.

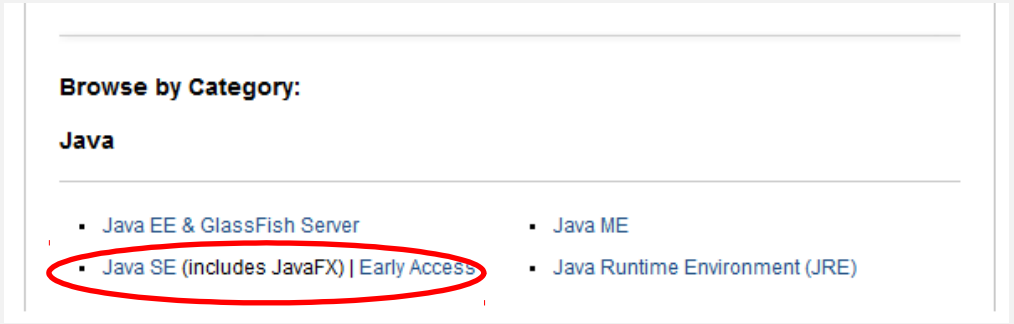
Mój program w Javie nie kompiluje się ⌐╦╦ Instalacja JVM, JRE, JDK

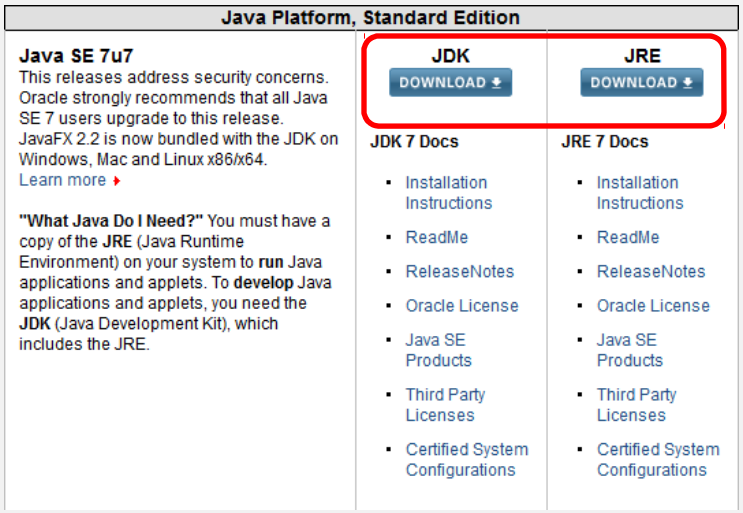
- JVM – *(ang. Java Virtual Machine),* tj. Wirtualna Maszyna Javy. Aby uruchomić aplikację napisaną przy pomocy języka Java, należy skompilować jej kod źródłowy. Kompilacja przebiega przy pomocy kompilatora i polega na przetłumaczeniu programu napisanego w Javie na kod wykonywalny, tzw. Bytecode (nie jest to ciąg instrukcji procesora komputera). JVM nie występuje jako samodzielny byt, a jedynie jako pewna część JRE lub JDK.
- JRE – *(ang. Java Runtime Environment)*, tj. Środowisko Uruchomieniowe Javy. Jest to JVM + zestaw klas i narzędzi niezbędnych do uruchamiania programów napisanych w Javie.
- JDK – *(ang. Java Development Kit)*, tj. Pakiet Programisty Javy. Jest to JRE + narzędzia niezbędne do implementacji i kompilacji aplikacji napisanych w języku Java.

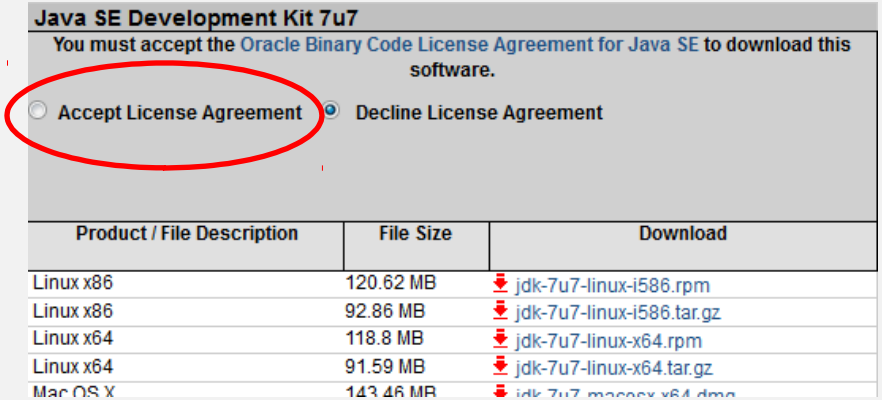
Instalacja JRE, JDK

- Odwiedzamy stronę producenta (obecnie Oracle) – **oracle.com**, po czym przechodzimy do działu Downloads.
- Na nowo otwartej stronie w obszarze Java wybieramy JavaSE (w zależności od potrzeb).
- Klikamy na DOWNLOAD dla JDK, bądź JRE.
- Akceptujemy licencję, ściągamy, istalujemy i...
- Enjoy ^^


ORACLE
Sign In/Register for Account
Help
PRODUCTS AND SERVICES
SOLUTIONS
DOWNLOADS
STORE
SUPPORT
Oracle Technology Network > Indexes > Download Index


Browse by Category:
Java
Java EE & GlassFish Server
Java SE (includes JavaFX) | Early Access
Java ME
Java Runtime Environment (JRE)


Java Platform, Standard Edition
Java SE 7u7
This releases address security concerns. Oracle strongly recommends that all Java SE 7 users upgrade to this release. JavaFX 2.2 is now bundled with the JDK on Windows, Mac and Linux x86/x64.
Learn more
"What Java Do I Need?" You must have a copy of the JRE (Java Runtime Environment) on your system to run Java applications and applets. To develop Java applications and applets, you need the JDK (Java Development Kit), which includes the JRE.
JDK
DOWNLOAD
JRE
DOWNLOAD
JDK 7 Docs
Installation Instructions
ReadMe
ReleaseNotes
Oracle License
Java SE Products
Third Party Licenses
Certified System Configurations
JRE 7 Docs
Installation Instructions
ReadMe
ReleaseNotes
Oracle License
Java SE Products
Third Party Licenses
Certified System Configurations

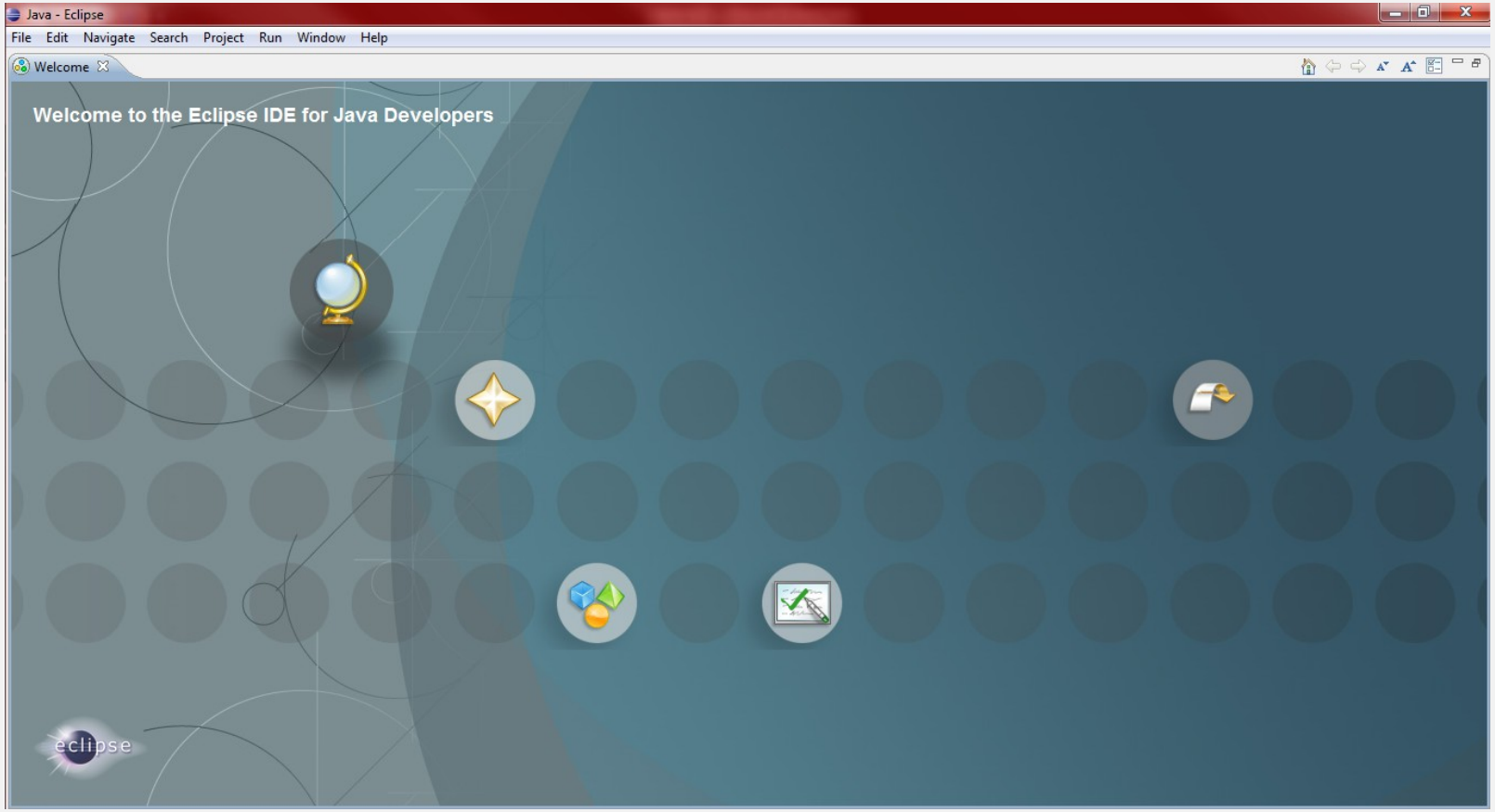

Java SE Development Kit 7u7
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.
Accept License Agreement
Decline License Agreement
Product / File Description
File Size
Download
Linux x86 120.62 MB jdk-7u7-linux-i586.rpm
Linux x86 92.86 MB jdk-7u7-linux-i586.tar.gz
Linux x64 118.8 MB jdk-7u7-linux-x64.rpm
Linux x64 91.59 MB jdk-7u7-linux-x64.tar.gz
Mac OS X 143.46 MB

Pierwsze starcie

- Każdorazowo przy uruchomieniu Eclipse, wraz z ekranem powitalnym pojawi się okno, w którym wybieramy workspace. We wskazanym miejscu będą przechowywane nasze projekty (domyślnie: *Moje Dokumenty\workspace*).
- Aby uniknąć tego kroku przy kolejnych uruchomieniach, zaznaczamy opcję

  „Use this as the default and do not ask again”

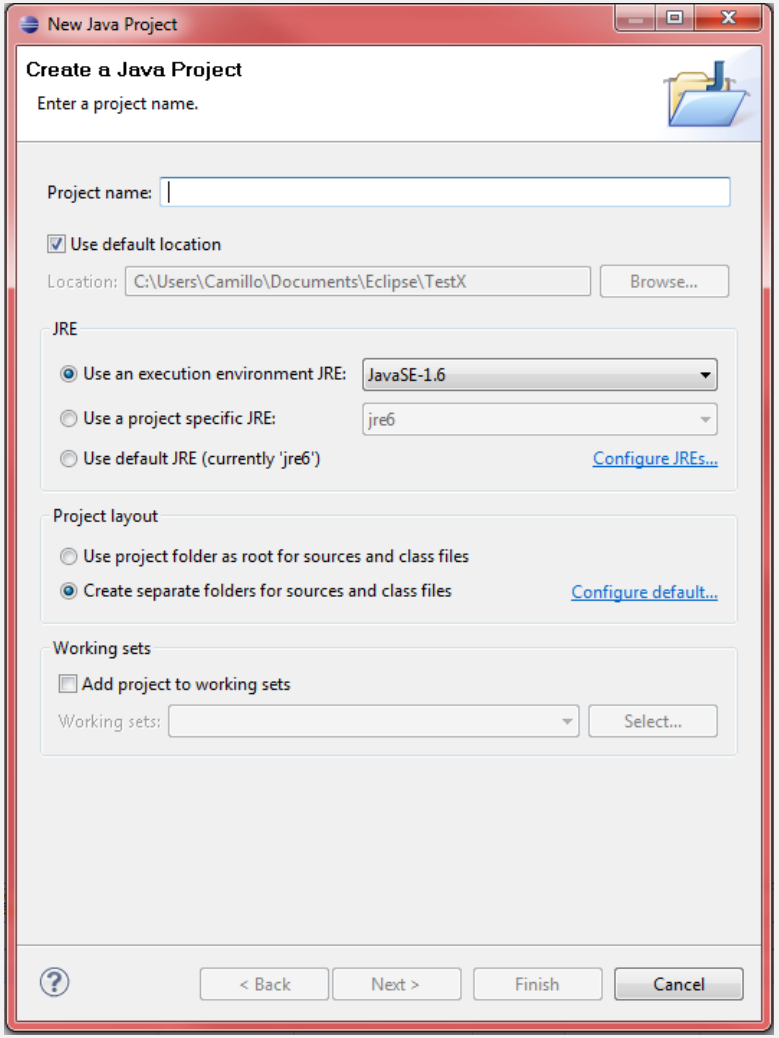
- Na ekranie powitalnym widnieją ikony odsyłające nas m.in. do tutoriali, wiadomości itd.



- Po przejściu do głównej prespektywy stykamy się z wieloma skonfigurowanymi widokami. Każde pojedyncze „okienko” wewnątrz Eclipse jest widokiem. Każdy z widoków przeznaczony jest do różnych zadań (tworzenie kodu, wyszukiwanie błędów itd.).
- Zmian w perspektywie dokonuje się poprzez przejście do zakładki *Window/Show View* i wybranie interesującego nas widoku.
- Perspektywę możemy oczywiście modyfikować przeciągając poszczególne widoki w wygodne dla nas miejsca *(mechanizm drag and drop)*.
- Możemy również przechodzić pomiędzy perspektywami wybierając kolejno Window/Open Prespective.

Pierwszy projekt

- W celu utworzenia projektu wybieramy kolejno *File/New/Java Project (*lub *Alt+Shift+N* i wybieramy *Java Project), po czym u*każe się okno kreatora projektu.
- Nadajemy projektowi nazwę, określamy lokalizację oraz wersję JRE.



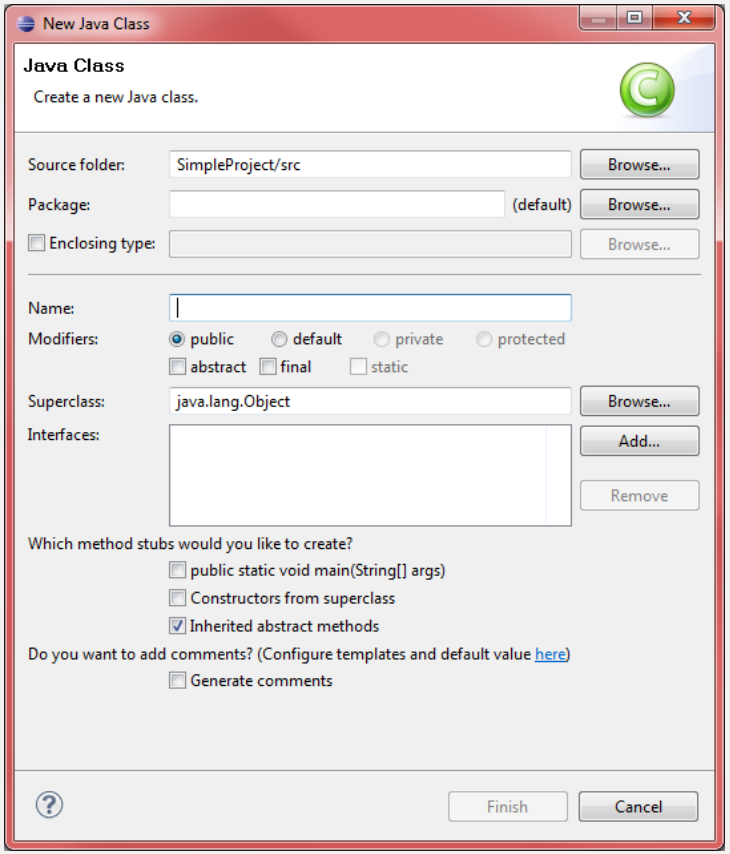
Nadejszła wiekopomna chwila...

„Kochaj albo rzuć” Kazimierz Pawlak

Pierwszy program w Javie.

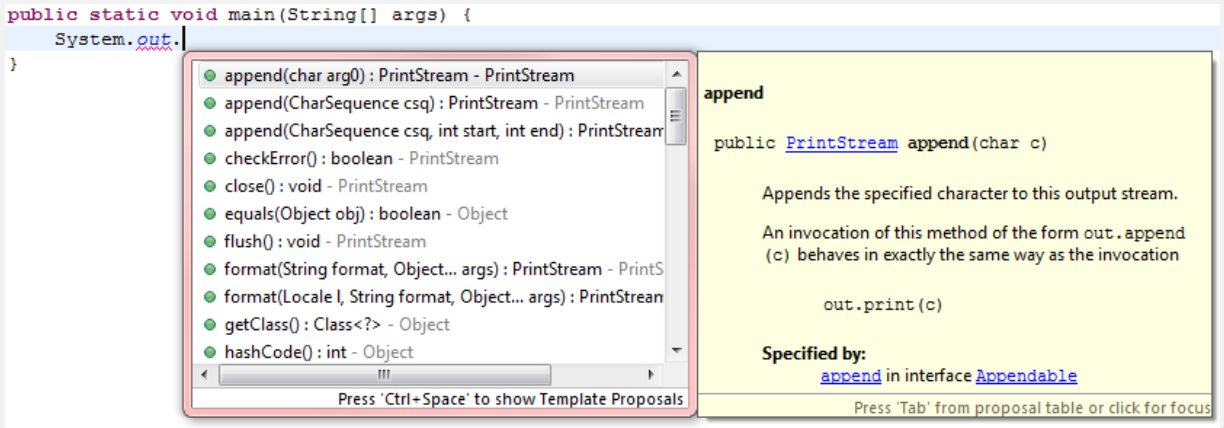
- Tworzymy nową klasę klikając ppm na projekcie i wybieramy *New/Class.*
- W oknie kreatora

Projektu znajduje się wiele cech, które możemy z góry nadać nowej klasie, m.in. określenie modyfikatorów dostępu oraz właściwości, czy też automatyczne wygenerowanie metody *main*.



- Po zaakceptowaniu wprowadzonych danych, w widoku kodu pojawia się skromny kod z publiczną klasą oraz metodą main().
- W celu utworzenia pierwszego, słynnego w każdym języku, programu piszemy wewnątrz metody main następującą składnię: *System.out.println(„Hello Eclipse”);*
- Zamiast pisać tak długą nazwę można posłużyć się skrótem: *„syso” po czym wcisnąć Ctrl+Space.*
- Aby skompilować program wybieramy *Run/Run* lub *Ctrl+F11*
- U dołu ekranu (domyślnie) w oknie konsoli pojawia się napis *Hello Eclipse.*

- Podczas pisania pierwszego programu nie sposób nie zauważyć *Asystenta wprowadzenia*.

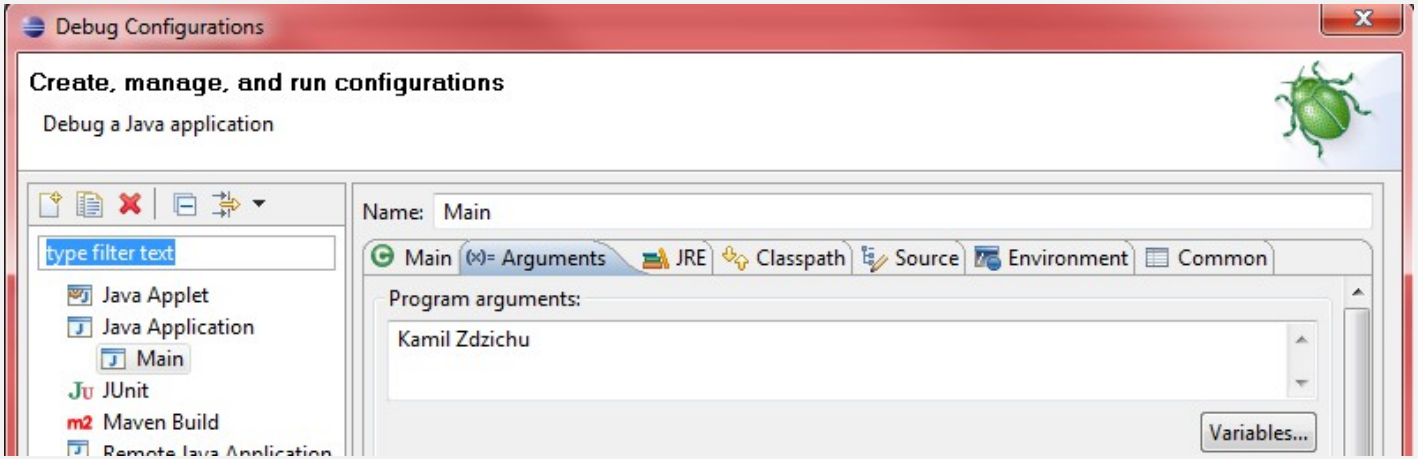


- Dzięki asystentowi wprowadzenia na bieżąco uzyskujemy podpowiedzi, gdy nie wiemy co napisać. Przedstawia on wszystkie dostępne z danego poziomu metody i pola na rzecz danego obiektu, do którego się odwołujemy, jak również opis dla wskazywanego elementu.

- Jeżeli podczas pisania popełnimy błąd (literówkę) asystent automatycznie znika. Aby go przywrócić wciskamy kombinację klawiszy *Ctrl+Space.*
- Na bieżąco sprawdzana jest poprawność kodu. Jeśli IDE uzna coś za błąd, automatycznie to podkreśli czerwoną linią (najczęściej są to *CompilException*). Drobne literówki, np. podczas wywoływania metody również zostaną podkreślone, lecz Eclipse sam zaproponuje właściwą nazwę, bądź utworzenie elementu o podanej nazwie.
- Nie wszystkie błędy zostaną wykryte przez Eclipse, jak np. błędy czasu wykonania *(ang. Runtime Exceptions)* m.in. Dzielenie liczby całkowitej przez zero *(ArithmeticException),* czy też próba otwarcia nieistniejącego pliku *(FileNotFoundException).*

Konfiguracja uruchomieniowa

- Jeżeli nasz program przyjmuje parametry z linii poleceń, a nie chcemy korzystać z konsoli systemowej, możemy wykorzystać w tym celu Eclipse.
- Wybieramy kolejno *Run/Debug Configurations...* Pojawi się okno:

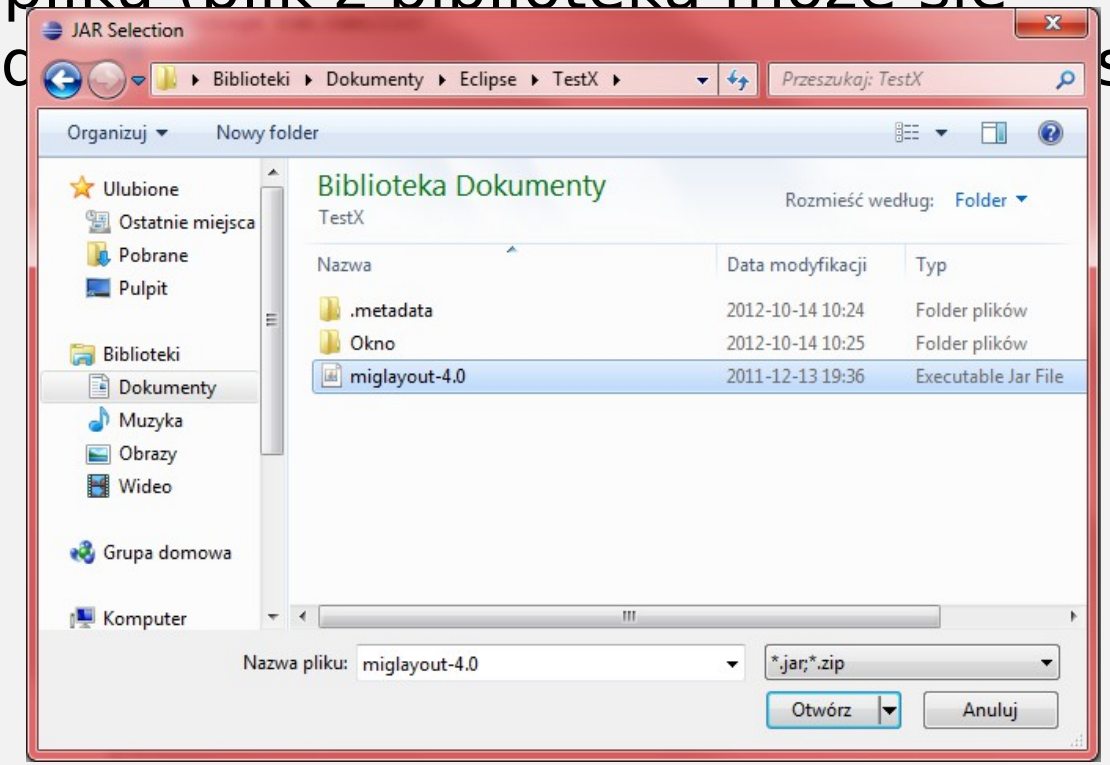


- W zakładce *Arguments*, podajemy parametry dla naszego programu.

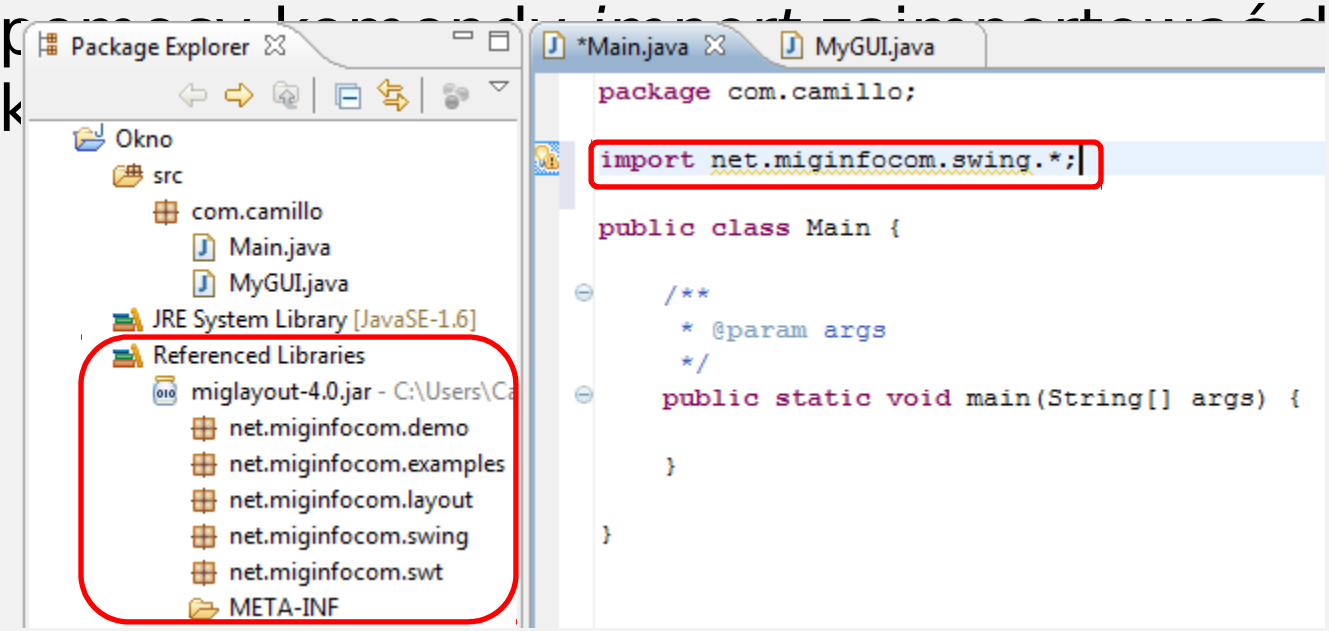
Dołączanie bibliotek

- Nie wszystkie dostępne dla Javy klasy znajdują się w standardowej bibliotece Javy. Istnieje bardzo duża liczba (wspieranych bądź nie przez Oracle) bibliotek, które bardzo łatwo można dołączyć do swojego projektu.
- Aby tego dokonać należy po pierwsze pobrać bibliotekę. Przykładem może być MigLayout – układ pozwalający na szybkie i proste rozmieszczenie elementów graficznych z biblioteki Swing (przyciski, pola tekstowe itp.) w oknie aplikacji.

- Aby dodać tę bibliotekę do projektu należy kliknąć ppm na projekcie i wybrać *Build Path/Add External Archives...*
- Ukaże się standardowe okno do dołączania plików, w którym szukamy interesującego nas pliku (plik z biblioteka może sie znajdować gdzieś na dysku).



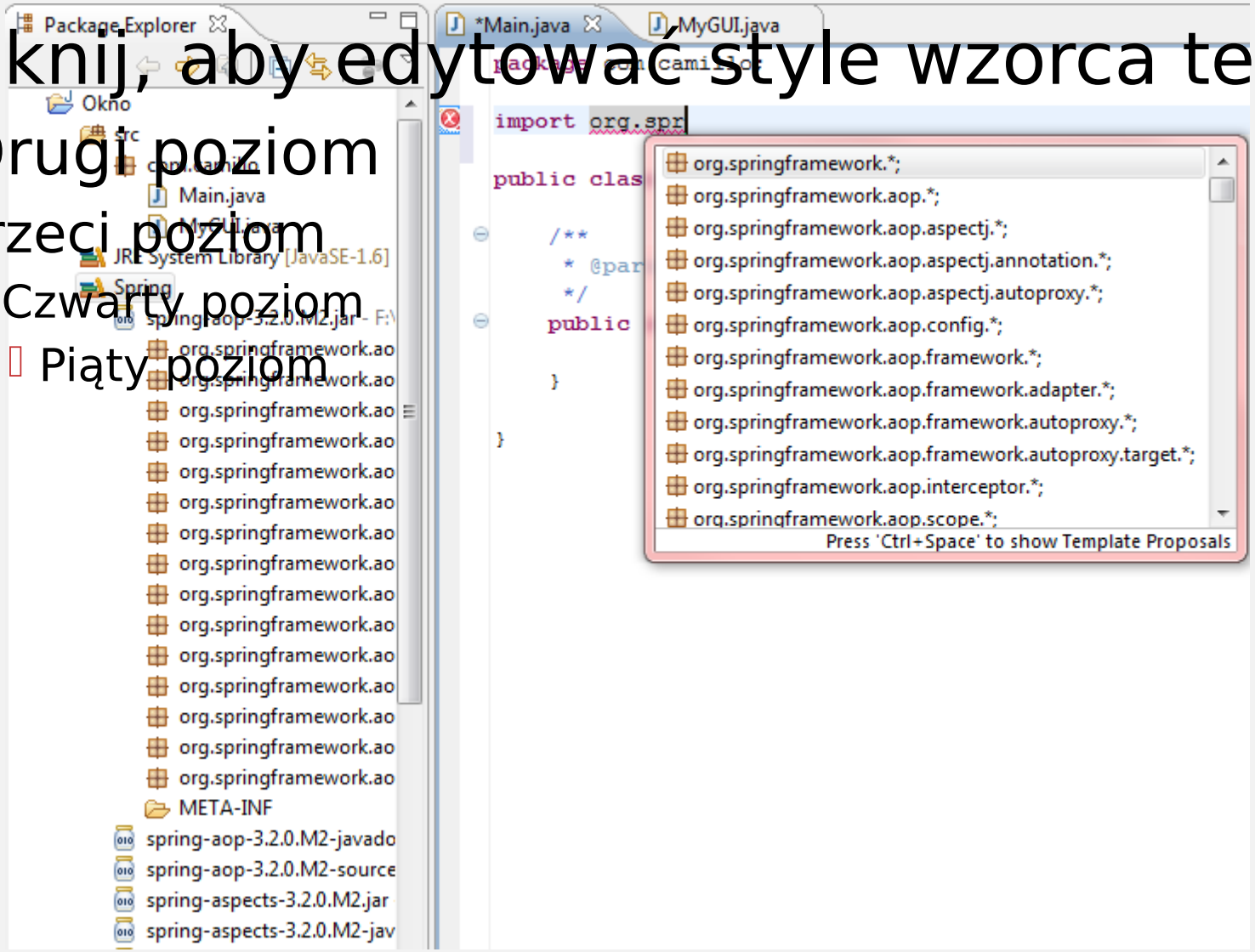
- Po dodaniu biblioteki w naszym projekcie utworzy się miejsce z referencjami do bibliotek *(Referenced Libraries)*, a w nim nasza biblioteka.
- Teraz aby skorzystać z tego co w danej bibliotece się znajduje wystarczy przy pomocy komendy *import* zaimportować do k



- Innym sposobem jest wybranie kolejno: Ppm na projekcie, *Build Path/Add Libraries…*
- W nowym oknie wybieramy *User Library* i klikamy *Next.*
- Jeśli nie ma wcześniej stworzonych przez nas bibliotek wybieramy *User Libraries…/New…*
- Nadajemy nazwę dla biblioteki, np. Spring i klikamy OK.
- Teraz klikamy na *Add JARs…*i wybieramy te pliki JAR, które są nam potrzebne. Po wszystkim klikamy OK i Finish.
- W ten sposób, mamy własną bibliotekę, oddzielną dla Framework’u Spring.

Kliknij, aby edytować style wzorca tekst

- Drugi poziom
- Trzeci poziom
  - Czwarty poziom
    - Piąty poziom

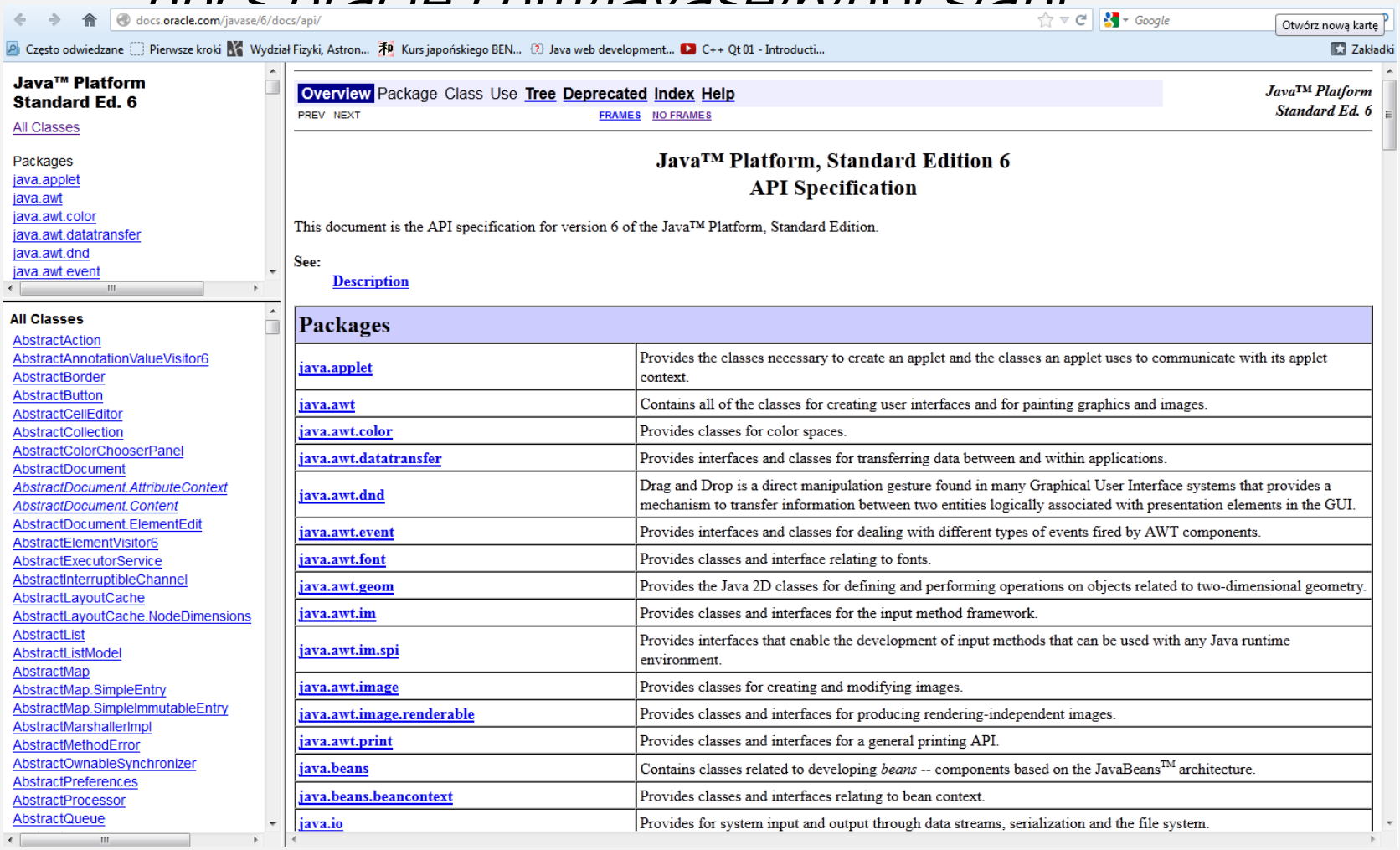


- Tego typu operacje możemy dokonywać już w momencie tworzenia projektu.
- Jeśli jednak okaże się, że biblioteka jest niepotrzebna, można ją usunąć w następujący sposób:
  - Klikamy ppm np. na projekcie i wybieramy *Build Path/Configure Build Path...*
  - W zakładce *Libraries* zaznaczamy bibliotekę i klikamy na przycisk *Remove*.

Javadoc

- Javadoc jest narzędziem ułatwiającym tworzenie dokumentacji technicznej projektu.
- Generuje dokumentacje z kodu źródłowego do kodu html, automatycznie dołączając informacje o nazwach komentowanych klas, pól, metod itd.
- Komentarze do przetworzenia przez javadoc muszą zaczynać się znakami /** a kończyć */ .
- Między znakami /** a */ można umieszczać dowolny kod html, który zostanie przeniesiony do dokumentacji.
- Javadoc rozpoznaje znaczniki dokumentacyjne zaczynające się od znaku @.

- Przykładem dokumentu javadoc jest chociażby dokumentacja Javy: *docs.oracle.com/javase/6/docs/api*



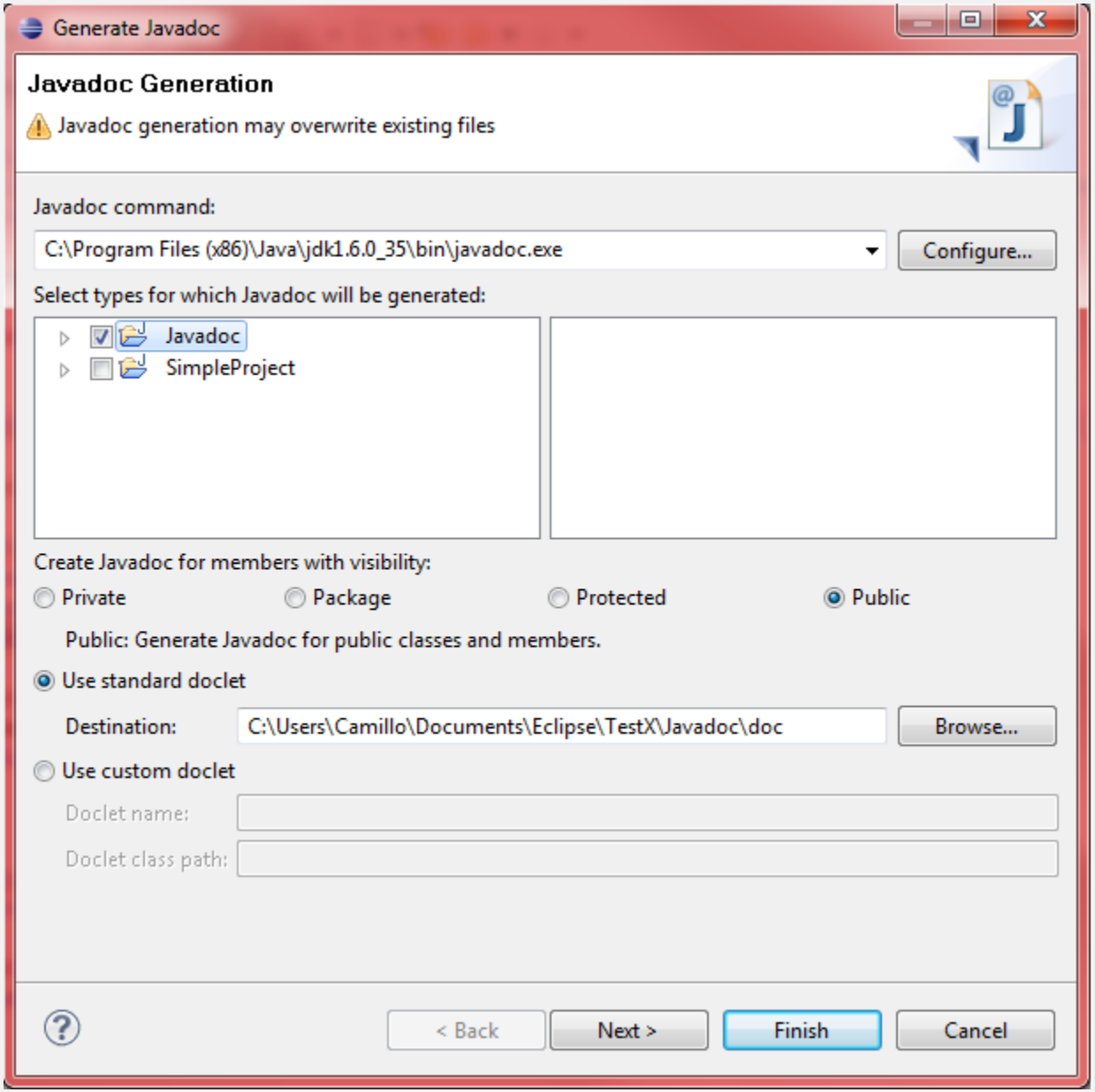
Javadoc w Eclipse

- Kliknij, aby edytować style wzorca teks
  - Drugi poziom
  - Trzeci poziom
    - Czwarty poziom
      - Piąty poziom

```
/**
 * Employee - klasa Employee dziedziczy klasy Person
 * @author Camil <a href="mailto:kamil.bieszczad@gmail.com">kamil.bieszczad@gmail.com</a>
 * @version alpha
 */
public class Employee extends Person {
    /* zmienna dotyczaca wysokości wypłaty danego pracownika */
    private float salary;
    /**
     * Konstruktor domyślny
     */
    public Employee() {}
    /**
     * Konstruktor klasy Employee ustawiający pola name, age oraz salary
     * @param name
     * @param age
     * @param salary
     */
    public Employee(String name, int age, int salary) {
        super(name, age);
        this.salary = salary;
    }

    /**
     * Zwraca wyplate pracownika
     * @return salary
     */
    public float getSalary() {
        return salary;
    }
```

- Aby wygenerować dokumentację należy wykonać następujące czynności:
  - Wchodzimy do *Project/Generate Javadoc…*
  - Ukaże się okno, gdzie w polu *Javadoc command* nameży wskazać występowanie pliku *javadoc.exe* (domyślnie w folderze zainstalowanego wcześniej jdk).
  - Należy wskazać projekt, dla którego zostanie wygenerowany Javadoc.
  - Należy również wybrać, czy chcemy wygenerować Javadoc tylko dla klas publicznych i ich elementów (opcja Public), czy też np. dla wszystkich klas, w tym też prywatnych (opcja Private lub Package).
  - Wybieramy lokalizację do zapisu i klikamy *Finish.*


Generate Javadoc
Javadoc Generation
Javadoc generation may overwrite existing files
Javadoc command:
C:\Program Files (x86)\Java\jdk1.6.0_35\bin\javadoc.exe
Configure...
Select types for which Javadoc will be generated:
Javadoc
SimpleProject
Create Javadoc for members with visibility:
Private
Package
Protected
Public
Public: Generate Javadoc for public classes and members.
Use standard doclet
Destination:
C:\Users\Camillo\Documents\Eclipse\TestX\Javadoc\doc
Browse...
Use custom doclet
Doclet name:
Doclet class path:
< Back
Next >
Finish
Cancel

- Jeśli podczas tworzenia dokumentacji będą jakieś błędy, kompilator na pewno nas o tym powiadomi w widoku *Console*.
- Domyślnie w widoku Package Explorer w folderze *doc* znajdują się wszystkie pliki w formacie HTML, zarówno do poszczególnych klas (np. Main.html), jak również do całości (index.html).
- Eclipse posiada wbudowaną przeglądarkę internetową, tak więc klikamy dwukrotnie na wybranym pliku lub ppm: *Open With/Web Browser*

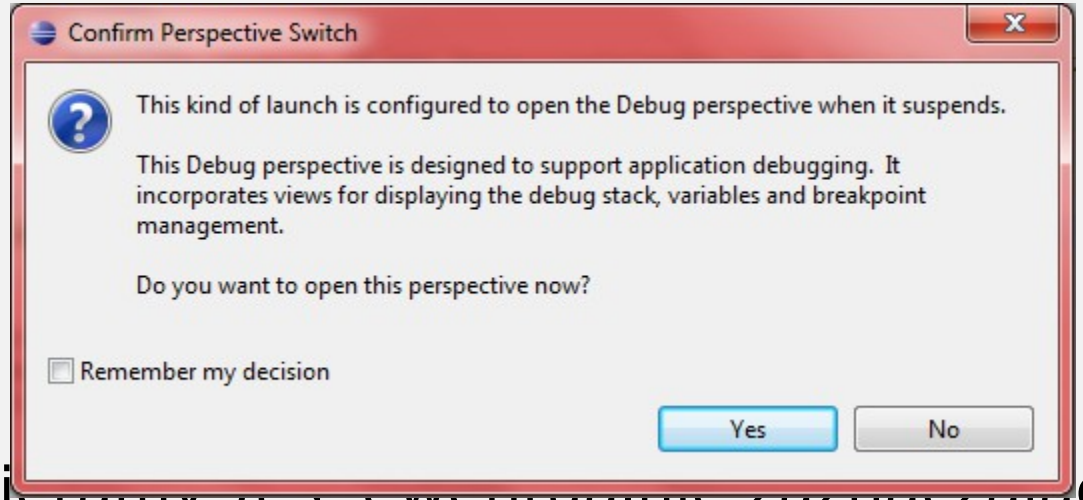
Debugger

- Debugger (*ang. odpluskwiacz*) – jest to program komputerowy, dzięki któremu można odnaleźć i zidentyfikować błędy (w innych programach), zwanych bugami(robakami).
- Dzięki debuggerom można efektywnie śledzić wartości poszczególnych zmiennych, wstrzymywać działanie programu w określonych miejscach, czy też wykonywać instrukcje krok po kroku.

Breakpoints – punkty wstrzymań

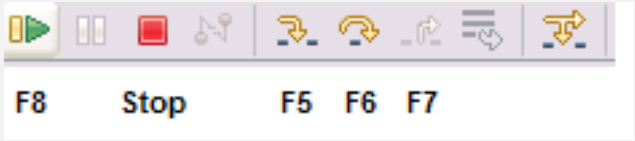
- W Eclipse dostępnych jest kilka typów wstrzymań linii:
  - punkty wstrzymań linii
  - punkty wstrzymań pól
  - punkty wstrzymań metod
  - punkty wstrzymań klasy
  - punkty wstrzymań wyjątków

- Po ustawieniu przynajmniej jednego punktu wstrzymania uruchamiamy debugera.
- Aby tego dokonać wybieramy kolejno *Run/Debug* po czym pojawi się okno:



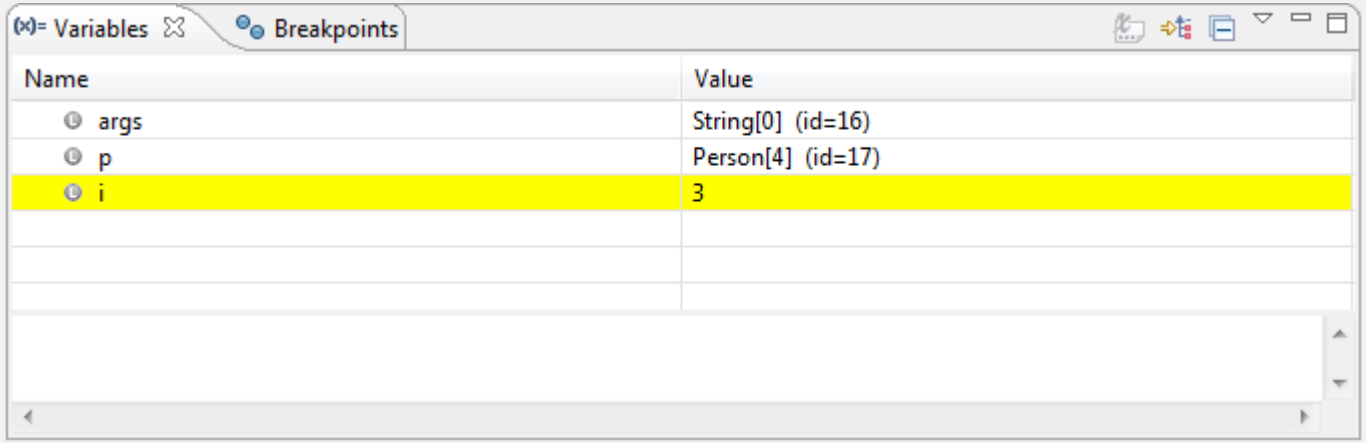
- Wybieramy *Yes*, ewentualnie zaznaczając *Remember my decision.*
- Zostaniemy przeniesieni do perspektywy *Debug.*

- W widoku *Debug* widnieją przyciski:



  - F5 – przejście do następnego kroku w programie. Jeśli następnym krokiem jest metoda, to polecenie wywoła ją, jednocześnie przechodząc do pierwszej linii jej wewnętrznego kodu.
  - F6 - przejście do następnego kroku w programie. Jeśli następnym krokiem jest metoda, to polecenie wywoła ją bez przejścia do jej wewnętrznego kodu.
  - F7 - Wykonuje do końca kod metody, w której znajduje się debug i wychodzi do metody wywołującej.
  - F8 - Opuszcza tryb debugowania krok po kroku. Przechodzi do następnego punktu wstrzymania.

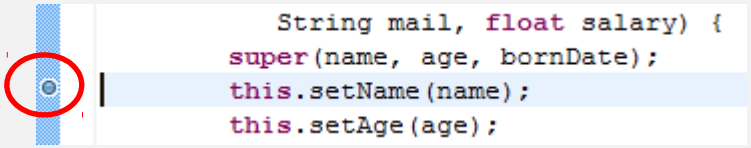
- Widok *Variables* – Zawiera listę dostępnych zmiennych wraz z ich wartościami.



- Możliwa jest również zmiana wartości zmiennych bezpośrednio w tabeli. Zmiany zatwierdza się klawiszem *Enter.*

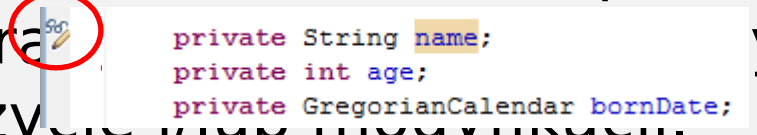
Punkty wstrzymań linii:

- Tego typu punktów wstrzymań używamy, gdy chcemy wstrzymać pracę naszego programu w konkretnej linii.



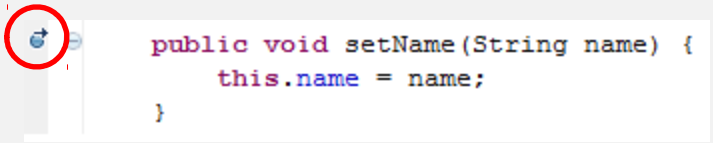
Punkty wstrzymań pól:

- Używane są gdy chcemy wstrzymać pracę naszego programu, gdy pole danej klasy jest wykorzystywane(odczyt/modyfikacja). Można również w *Breakpoint properties...* wybra yć aktywny przy odczycie i/lub modyfikacji.



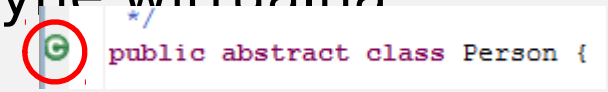
Punkty wstrzymań metod:

- Używamy ich gdy interesuje nas, kiedy program wchodzi i/lub wychodzi z danej metody. Te breakpointy ustawiamy na wysokości rozpoczęcia definicji metody. We właściwościach można określić, czy program ma się zatrzymać na wejściu i/lub wyjściu metody.



Punkty wstrzymań klas:

- Używana, gdy interesuje nas moment pierwszego ładowania danej klasy przez maszynę wirtualną

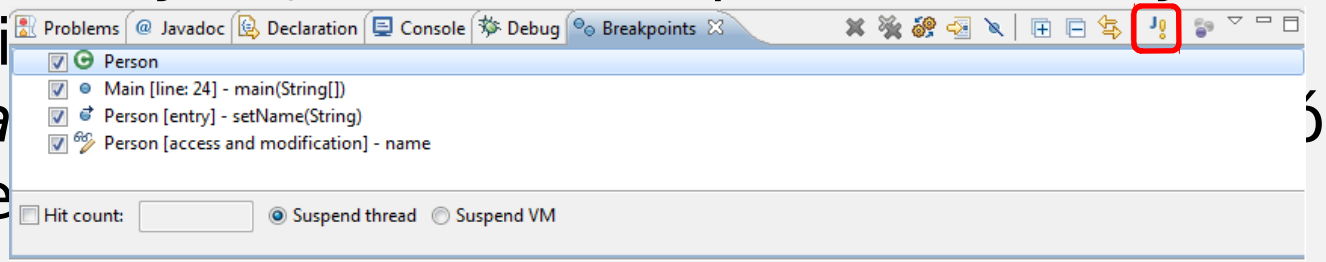


Punkty wstrzymań wyjątków:

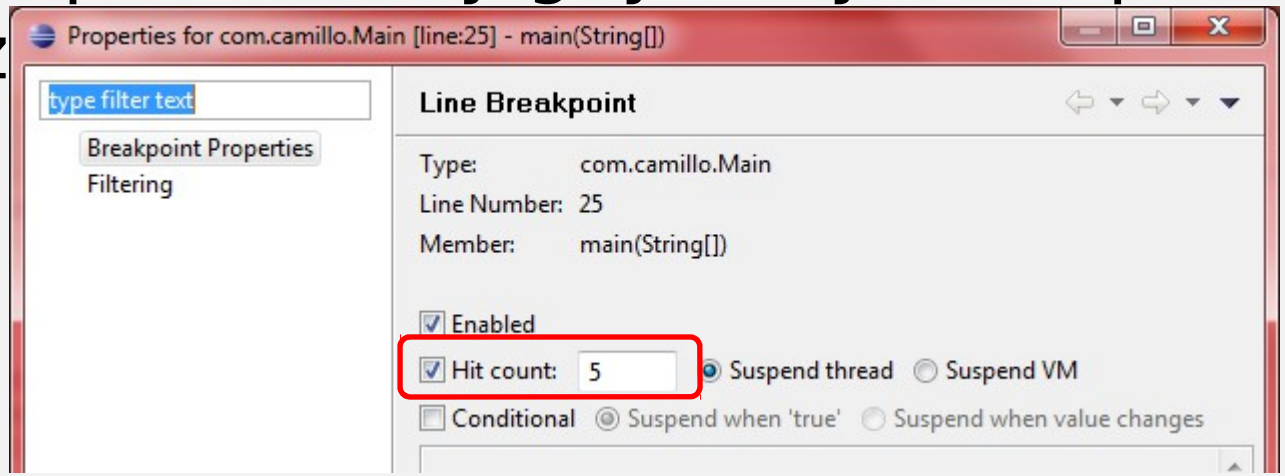
- Używany, gdy w programie występuje jakiś wyjątek, ale problemem jest określenie przyczyny oraz miejsca, w którym występuje.
- Otwieramy:

Window/Show View/Other/Debug/Breakpoints

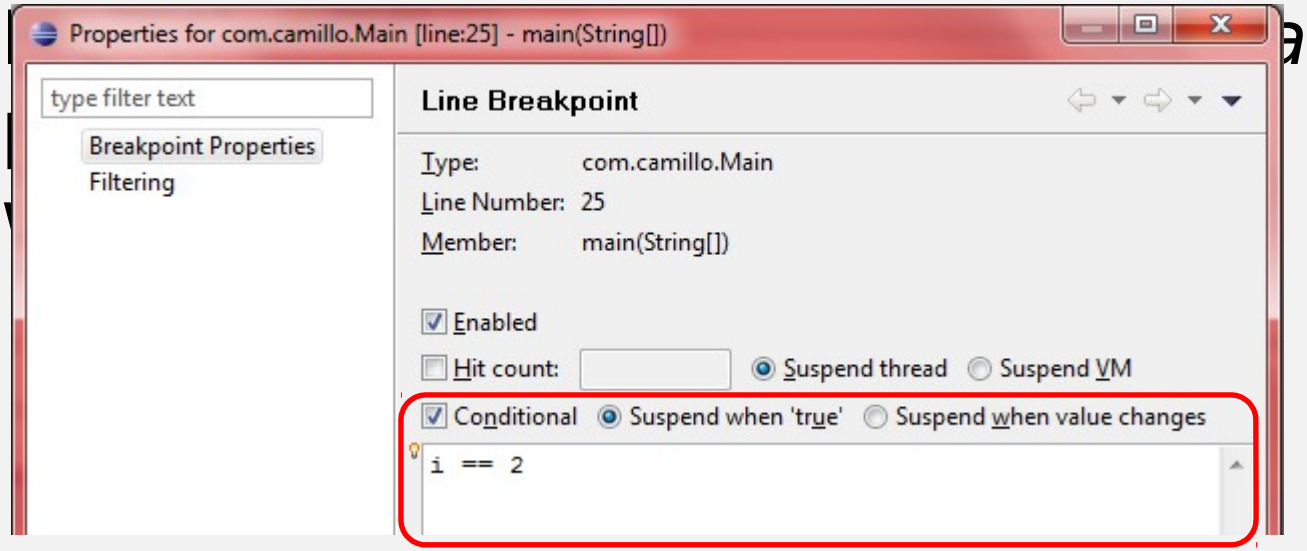
Otworzy się widok *Breakpoints*, w którym wybi
Brea ory
chce



- Właściwości punktów wstrzymań:
  - Hit Count - ilość trafień. Ustawiamy breakpoint, po czym w jego właściwościach zaznaczamy opcję *Hit count,* po czym podajemy wartość *N*. Opcja ta pozwala na wstrzymanie działania programu, dopiero wtedy gdy dany breakpoint z



- Condition – warunek. Jak nazwa wskazuje, punktom wstrzymań można nadawać warunki. Po spełnieniu danego warunku breakpoint uaktywni się, co spowoduje wstrzymanie działania programu.
- *al,*



- Dzięki widokowi *Breakpoints* możemy w szybki sposób:
  - dodawać punkty wstrzymań,
  - usuwać(pojedynczo lub wszystkie jednocześnie),
  - szybko przejść do miejsca występowania danego punktu wstrzymania,
  - dodać punkty wstrzymań wyjątków,
  - dezaktywować punkty wstrzymań.

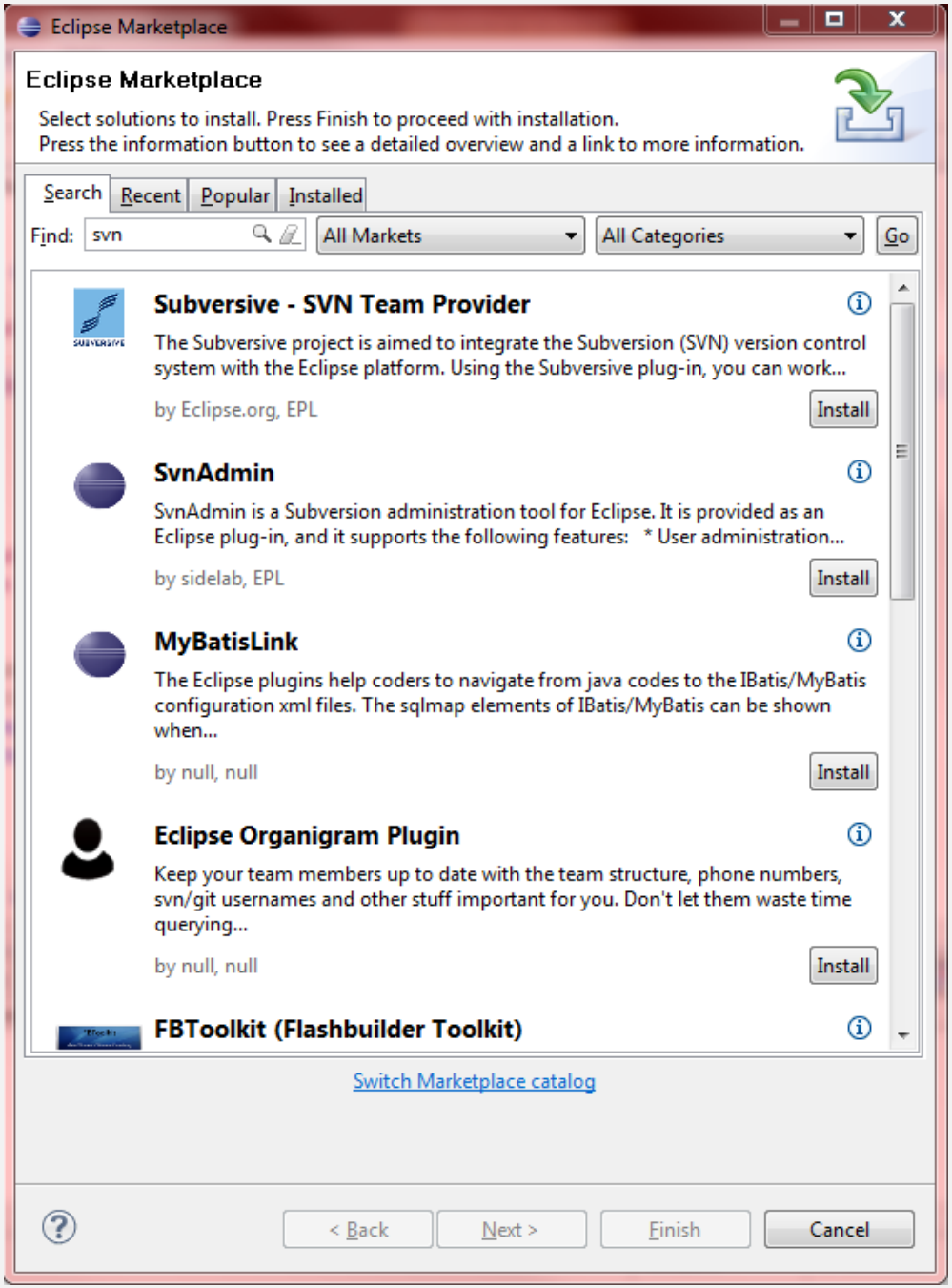
Tworzenie plików JAR

- Czym są pliki JAR?
- Ogólnie pliki JAR są archiwami, podobnymi do plików rar czy zip, z tym że zawierają w sobie program Javy.
- Charakterystyczną rzeczą plików JAR jest to, że oprócz plików skompilowanych klas czy też plików źródłowych zawierają w sobie informację o sposobie uruchomienia programu. Informacje te umieszczone są w pliku manifestu *(manifest.mf)* znajdującym się w katalogu META.

- Aby utworzyć plik JAR w Eclipse należy:
  - Kliknąć ppm na projekcie i wybrać *Export…*
  - Wybieramy opcję *Java/JAR file*
  - Wybieramy co ma być umieszczone w pliku JAR oraz określamy miejsce zapisu pliku.
  - Klikamy *Next* i akceptujemy domyślne ustawienia.
  - Zaznaczamy *Generate the manifest file* oraz określamy klasę, z której program będzie startował.
  - Klikamy *Finish* i gotowe ^^

Pobieranie wtyczek

- Eclipse jest bardzo dobrze rozbudowaną platformą, ale nie zawsze to czego potrzebujemy jest dostępne od razu po pierwszym uruchomieniu.
- Dla Eclipse są tworzone różnego rodzaju wtyczki (plugin).
- Aby je zainstalować, można skorzystać z *Eclipse Marketplace* dostępnym w *Help/Eclipse Maretplace...*
- W karcie *Search* w polu *Find* wpisujemy nazwę interesującej nas wtyczki, po czym klikamy na przycisk *Go.*
- Z listy wybieramy interesującą nas pozycję i klikamy *Install.*


Eclipse Marketplace
Eclipse Marketplace
Select solutions to install. Press Finish to proceed with installation.
Press the information button to see a detailed overview and a link to more information.
Search
Recent
Popular
Installed
Find:
svn
All Markets
All Categories
Go
Subversive - SVN Team Provider
The Subversive project is aimed to integrate the Subversion (SVN) version control system with the Eclipse platform. Using the Subversive plug-in, you can work...
by Eclipse.org, EPL
Install
SvnAdmin
SvnAdmin is a Subversion administration tool for Eclipse. It is provided as an Eclipse plug-in, and it supports the following features: * User administration...
by sidelab, EPL
Install
MyBatisLink
The Eclipse plugins help coders to navigate from java codes to the IBatis/MyBatis configuration xml files. The sqlmap elements of IBatis/MyBatis can be shown when...
by null, null
Install
Eclipse Organigram Plugin
Keep your team members up to date with the team structure, phone numbers, svn/git usernames and other stuff important for you. Don't let them waste time querying...
by null, null
Install
FBToolkit (Flashbuilder Toolkit)
Switch Marketplace catalog
< Back
Next >
Finish
Cancel

Podstawowe skróty

- *Ctrl+Space* – podgląd wszystkich dostępnych metod i pól, które można wywołać na rzecz danego obiektu. Również automatyczne dokończenie wpisywanej treści.
- *Ctrl+1* – w sposób automatyczny poprawia kod (ostrzeżenia i błędy).
- *Ctrl+/* – zakomentowanie aktualnej linii.
- *Ctrl+Shift+F* – auto formatowanie kodu.
- *Ctrl+D* – usuwa cały wiersz, w którym się znajdujemy.
- *Ctrl+M* – rozciągnięcie aktualnego widoku na cały ekran. Ten sam skrót do powrotu.
- *Ctrl+L* – idzie do linii o podanym numerze.
- *Ctrl+Q* – przenosi w miejsce ostatniej zmiany.
- *Ctrl+F6* – przechodzenie pomiędzy otwartymi edytorami.
- Alt+Shift+R – zmiana nazwy danego pola we wszystkich miejscach jego wystąpienia.
- *Ctrl+Shift+L* – pokazuje listę skrótów.

Porady i wskazówki

- Refractoring:
  - Wyodrębnienie zmiennej lokalnej:
  - Zaznaczamy „Hello Eclipse”
  - Wciskamy *Alt+Shift+L*
  - Podajemy nazwę dla zmiennej.

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

⇒

```
public static void main(String[] args) {
    String hello = "Hello Eclipse";
    System.out.println(hello);
    System.out.println(hello);
}
```

  - Działanie odwrotne: *Alt+Shift+I*
  - Wyodrębnienie metody:
  - Zaznaczamy *System.out.println(hello);*
  - Wciskamy Alt+Shift+M
  - Podajemy nazwę metody

```
public static void main(String[] args) {
    String hello = "Hello Eclipse";
    System.out.println(hello);
    System.out.println(hello);
}
```

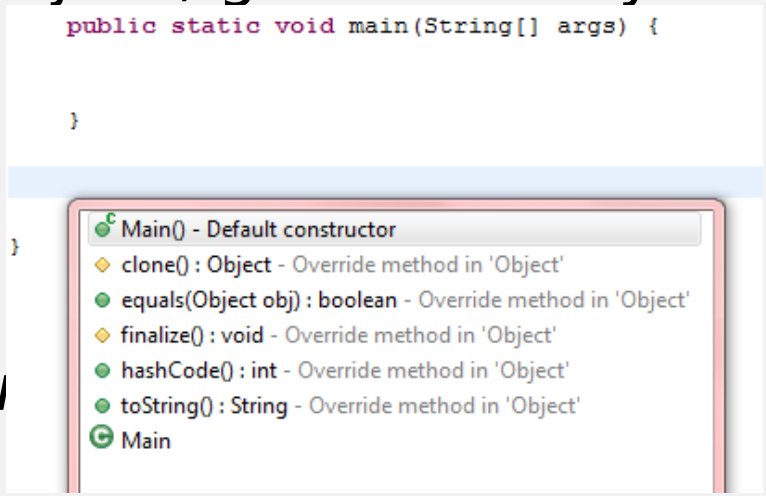
⇒

```
public static void main(String[] args) {
    String hello = "Hello Eclipse";
    syso(hello);
    syso(hello);
}

private static void syso(String hello) {
    System.out.println(hello);
}
```

- Domyślny konstruktor:
  - Ustawiamy kursor w miejscu, gdzie chcemy umieścić konstruktor,
  - Wciskamy *Ctrl+Space*
  - Wybieramy:

  Klasa() – Default constru

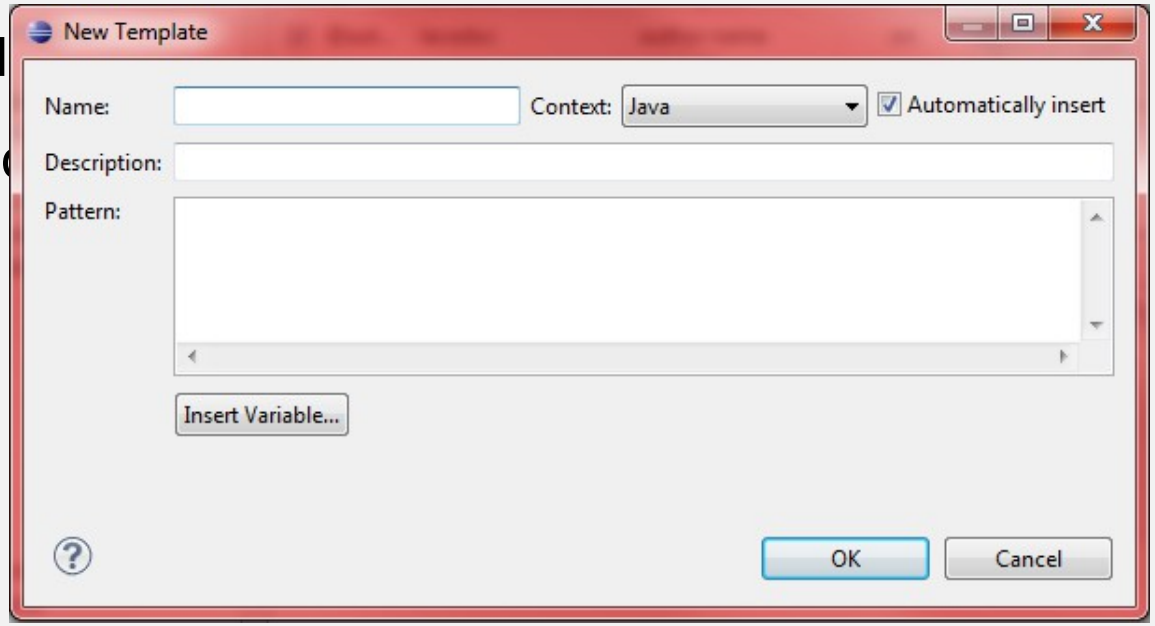


- Domyślne szablony kodu:
  - Przykładem domyślnego szablonu jest *pętla for.*
  - Aby użyć sza a następnie *Ctrl+Space.* N ów *pętli for*.

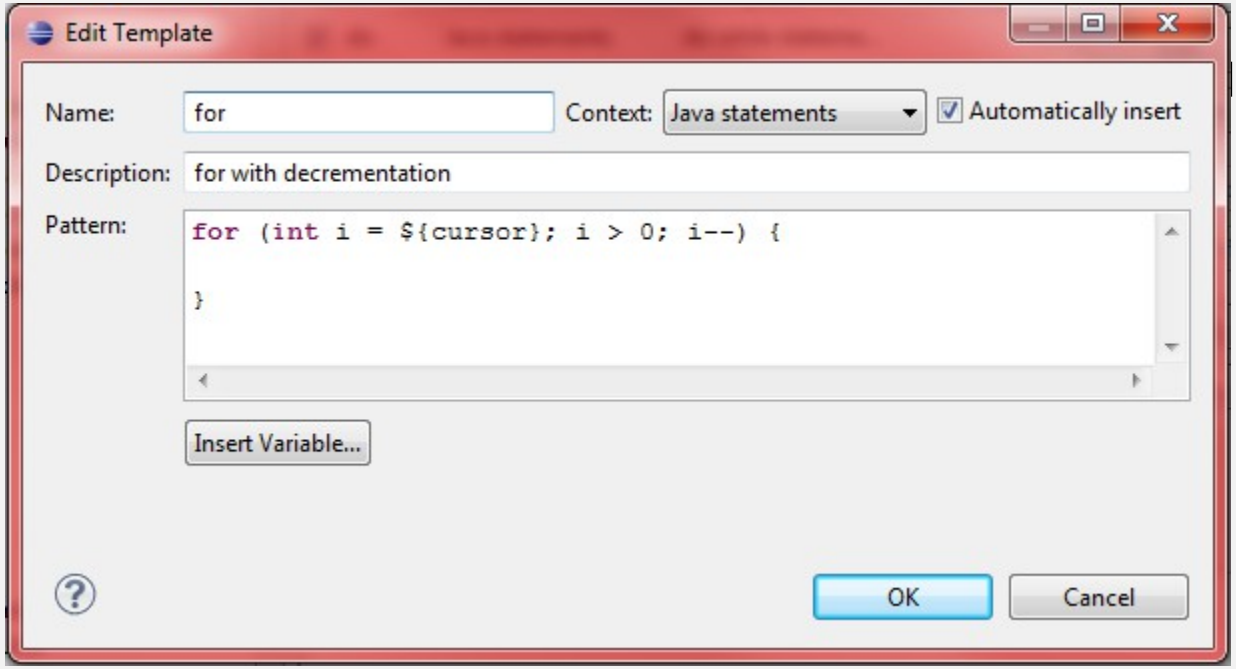
```
public static void main(String[] args) {
    for (int i = 0; i < args.length; i++) {

    }
}
```

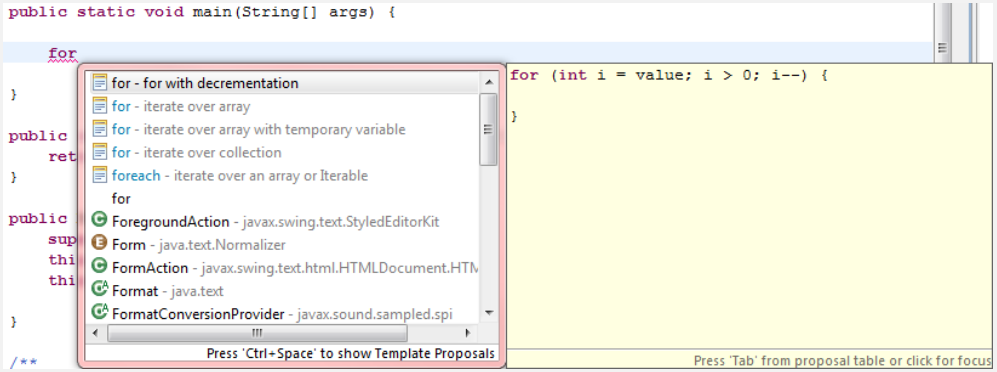
- Definiowanie własnego szablonu kodu:
  - Eclipse pozwala na definiowanie własnych szablonów, które przyspieszają pracę programisty oraz eliminują pojawienie się błędów.
  - Aby stworzyć własny szablon należy wybrać kolejno : *Window/Preferences/Java/Editor/Templates*
  - N
  - P



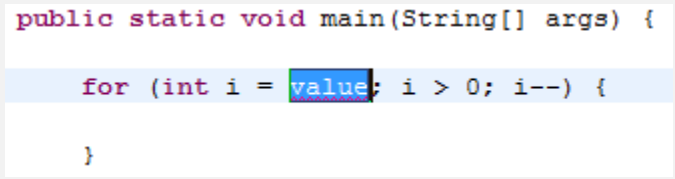
- W polu *Name:* podajemy nazwę szablonu.
- W polu *Description:* opis dla danego szablonu.
- Zarówno to co podamy w polu *Name* jak i w polu *Description*, będzie widoczne w asystencie wprowadzenia *(Ctrl+Space).*
- W , np.



- W szablonach zmienne ${...} zmieniane są na odpowiednie wartości.
  - ${cursor} – ustawia kursor w danym miejscu.
  - ${value} – wskazuje miejsce do nadania wartości.
  - ${enclosing_method} – nazwa aktualnej metody.
  - itd..
- Teraz wpisując nazwę danego szablonu i naciskając *Ctrl+Space* ujrzymy:



- A po wciśnięciu *Enter:*



- Generacja kodu:
  - Generowanie getterów i setterów.
  - Mając przygotowane pola, np. *String name, int age,* klikamy ppm w miejscu gdzie chcemy stworzyć dla tych pól gettery oraz settery.
  - Wybieramy *Source/Generate Getters and Setters…*
  - Wybieramy, dla których pól mają zostać wygenerowane gettery i settery bądź tylko je

```
    }

    private int age;
    private String name;


}
```

```
private int age;
private String name;

public int getAge() {
    return age;
}
public void setAge(int age) {
    this.age = age;
}
public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
```

- Generowanie konstruktorów:
  - W celu wygenerowania konstruktora przypisującemu wartości swoich parametrów odpowiednim polom klasy, klikamy ppm w miejscu, gdzie chcemy umieścić konstruktor.
  - Wybieramy *Source/Generate Constructor using Fields...*
  - a, które

```
}
private int age;
private String name;

public int getAge() {
    return age;
}
public void setAge(int age) {
    this.age = age;
}
```

```
}

public Main(int age, String name) {
    super();
    this.age = age;
    this.name = name;
}

private int age;
private String name;

public int getAge() {
    return age;
```

- Generowanie Javadoc:
  - Ustawiamy kursor na nazwie metody.
  - Wybieramy *Alt+Shift+J*

```
public double abs(double arg) {
    return arg > 0 ? arg : -arg;
}
```

```
/**
 * @param arg
 * @return
 */
public double abs(double arg) {
    return arg > 0 ? arg : -arg;
}
```

- Czyszczenie kodu

Eclipse pozwala nam na zmianę, czy też czyszczenie kodu. Polega on m.in. na:

dodawaniu niezaimplementowanych metod,

konwersji pętli typu for do foreach,

dodaniu lub usunięciu nawiasów w pętlach

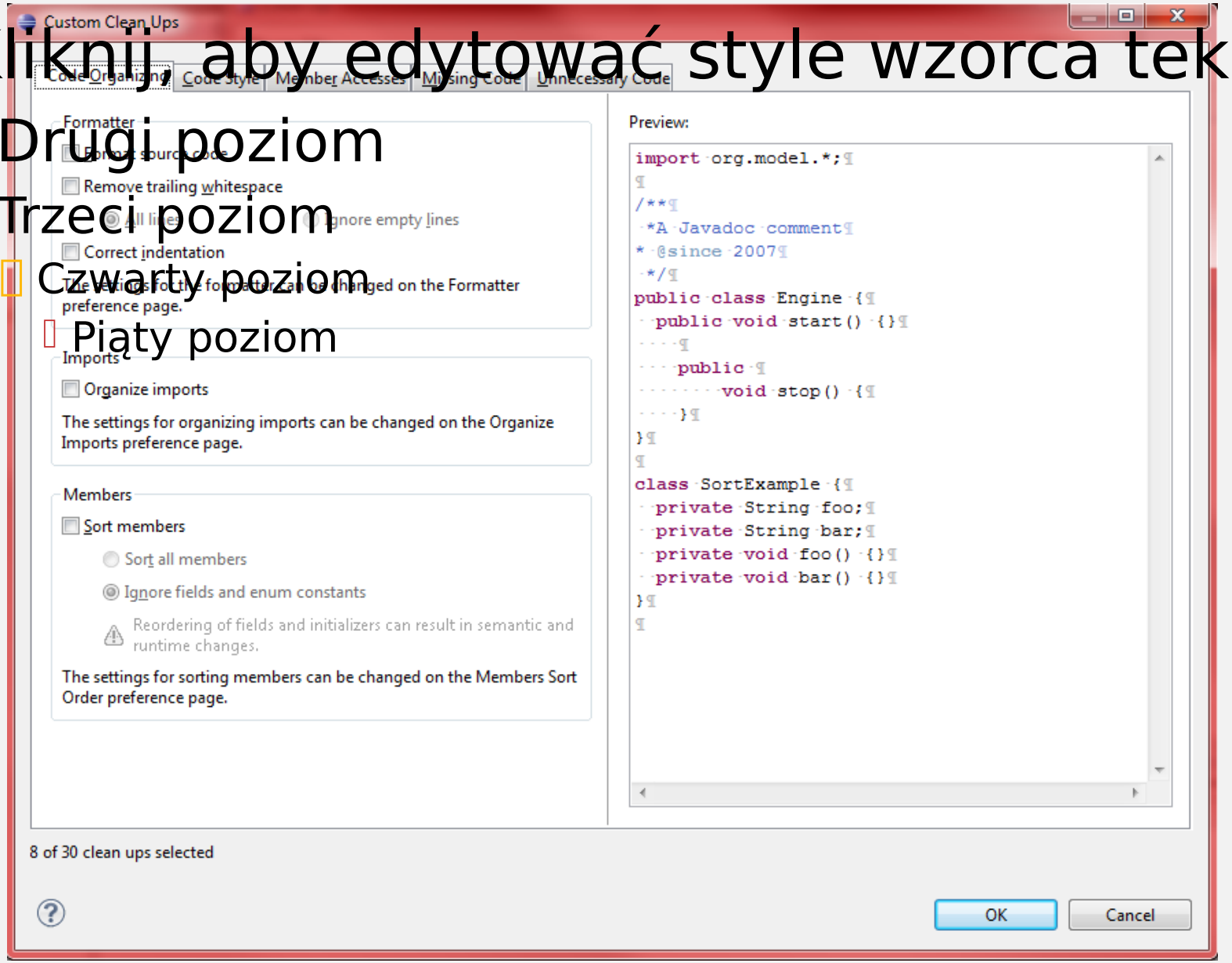
oraz wiele więcej przydatnych rzeczy.

Jeśli ktoś lubi porządek w swoim kodzie, warto się zapoznać z tą opcją.

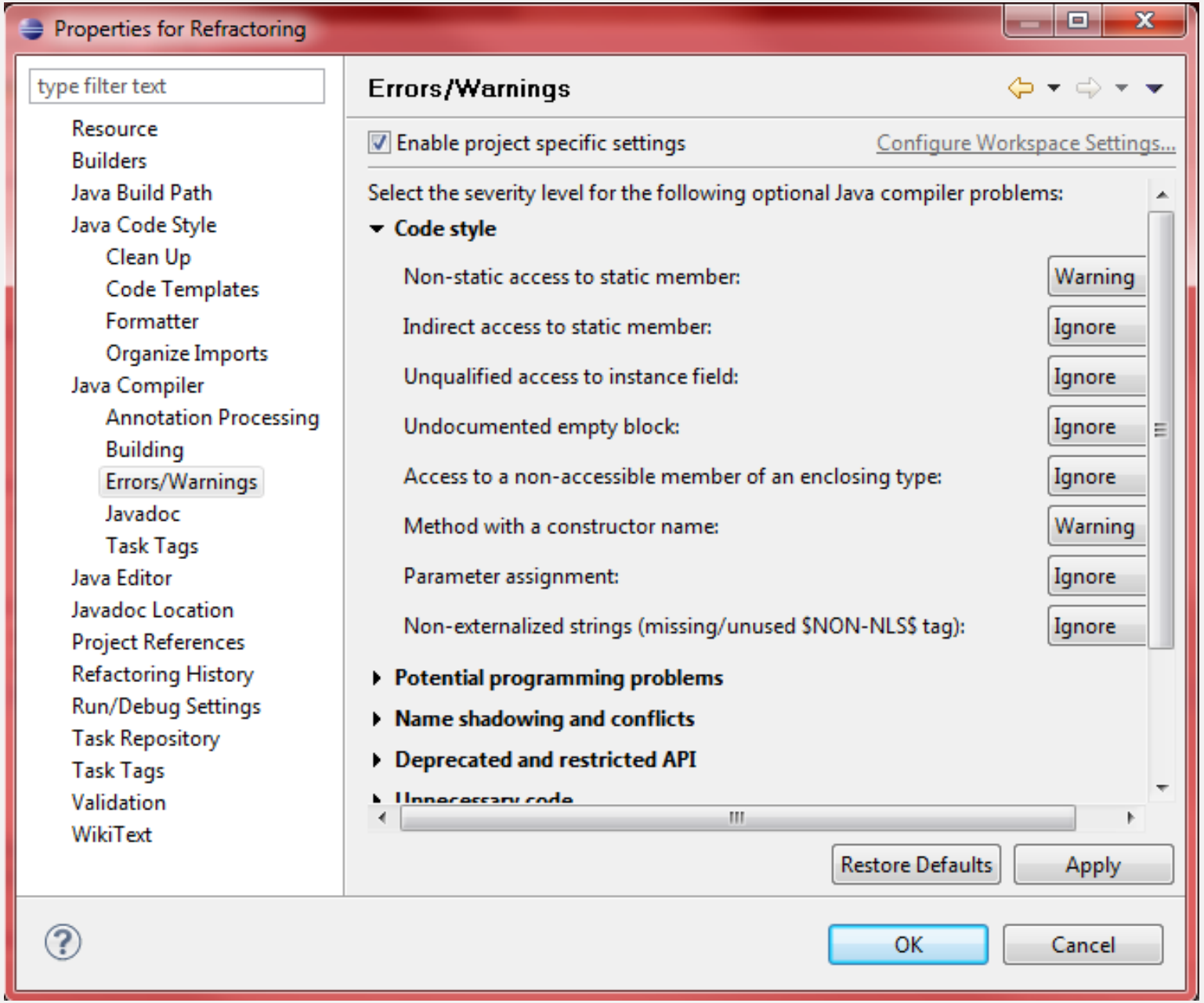
Aby z niej skorzystać wybieramy: *Source/Clean Up…,* po czym jeśli chcemy skomponować własny profil, wybieramy *Use custom profile*, a następnie *Configure*

Pojawi się okno z kilkoma kartami, a w nich wiele opcji do wyboru z podglądem po prawej stronie.

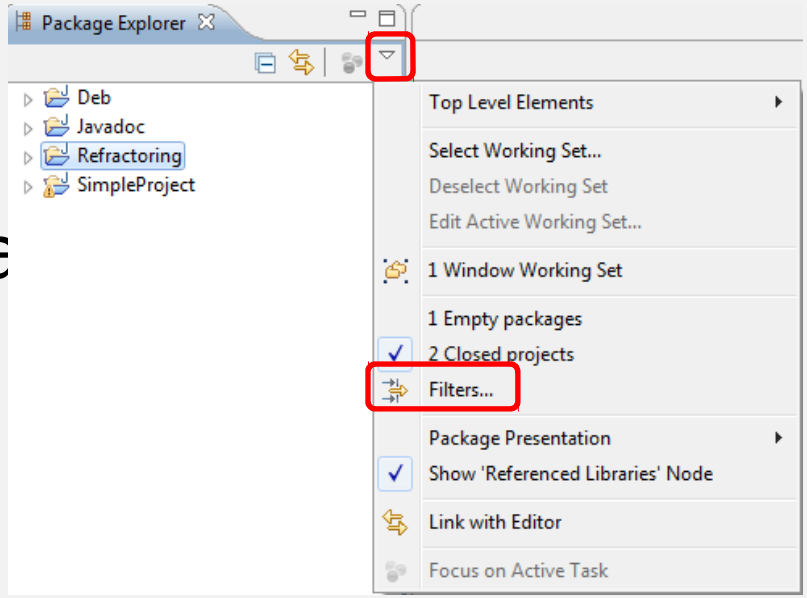
- Kliknij, aby edytować style wzorca teks
  - Drugi poziom
  - Trzeci poziom
    - Czwarty poziom
      - Piąty poziom



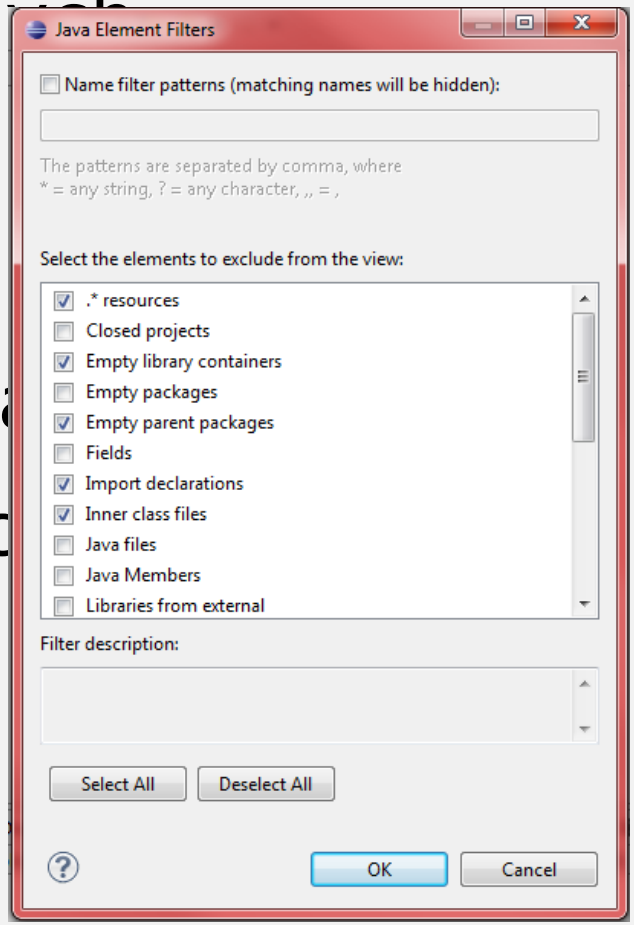
- Jeśli w grupie programistów pracujących nad danym projektem istnieją osoby, które nie chcą trzymać się pewnych standardów i uważają, że kod nie musi pięknie wyglądać, to jest sposób aby wymusić na takich osobach trzymanie się norm.
- Eclipse udostępnia zmianę ustawień kompilatora. Można np. poinformować kompilator, że metoda o takiej samej nazwie jak klasa jest niedozwolona, przez co będzie zgłaszany błąd.
- Aby tego dokonać klikamy ppm na projekcie i wybieramy *Properties/Java Compiler/ Errors/Wornings,* po czym zaznaczamy *Enable project specific settings.*
- W tym miejscu widnieje wiele opcji do wyboru, które możemy ustawić na ignorowane przez kompilator, jak też na te, przed którymi kompilator nas tylko ostrzeże oraz te, które kompilator ma uznawać za błąd.


Properties for Refactoring
type filter text
Resource
Builders
Java Build Path
Java Code Style
Clean Up
Code Templates
Formatter
Organize Imports
Java Compiler
Annotation Processing
Building
Errors/Warnings
Javadoc
Task Tags
Java Editor
Javadoc Location
Project References
Refactoring History
Run/Debug Settings
Task Repository
Task Tags
Validation
WikiText
Errors/Warnings
Enable project specific settings
Configure Workspace Settings...
Select the severity level for the following optional Java compiler problems:
Code style
Non-static access to static member:
Warning
Indirect access to static member:
Ignore
Unqualified access to instance field:
Ignore
Undocumented empty block:
Ignore
Access to a non-accessible member of an enclosing type:
Ignore
Method with a constructor name:
Warning
Parameter assignment:
Ignore
Non-externalized strings (missing/unused $NON-NLS$ tag):
Ignore
Potential programming problems
Name shadowing and conflicts
Deprecated and restricted API
Restore Defaults
Apply
OK
Cancel

- Filtry Package Explorer’a
  - Przy dużych zadaniach, w projekcie może pojawić się wiele elementów, które jak się okazuje, często są puste, jak również zawierają pliki o różnych rozszerzeniach i wiele innych.
  - Wchodząc do znajdziemy ta



- Jak widać, widnieje tu wiele elementów, tj. zamknięte projekty, puste kontenery bibliotek, puste pakiety, pola i wiele innych.
- Należy uważać, aby po operacji filtrowania nie zapomnieć o istniejących ale ukrytych elementach naszego projektu.



- Kliknij, aby edytować style wzorca teks
  - Drugi poziom
  - Trzeci poziom
    - Czwarty poziom
      - Piąty poziom