# Projektowanie aplikacji internetowych – laboratorium

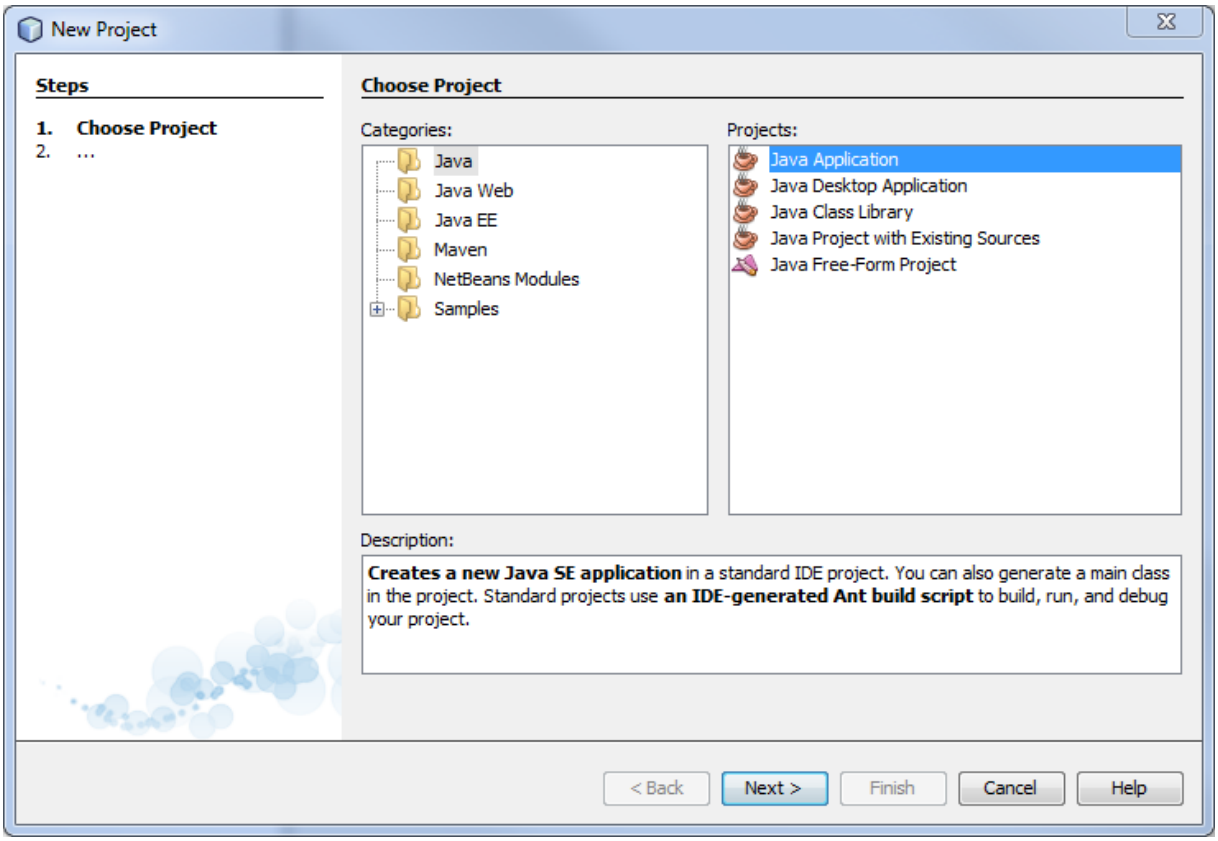
## *Programowanie w języku Java*

Do realizacji projektu potrzebne jest zintegrowane środowisko programistyczne NetBeans 7 (zrzuty ekranów pochodzą z wersji 7.0.1). Każda z aplikacji powinna stanowić odrębny projekt.

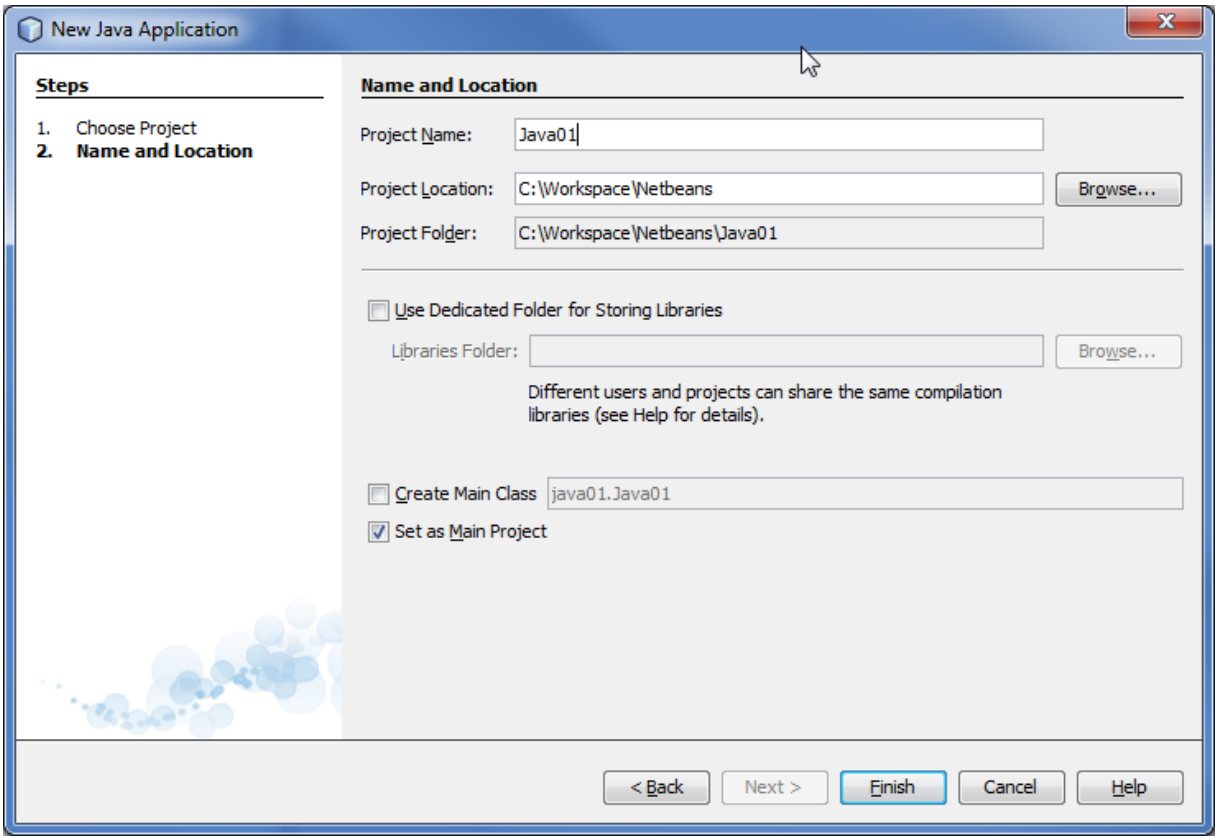
1. Twoim pierwszym zadaniem będzie zbudowanie najprostszej aplikacji konsolowej (bez graficznego interfejsu użytkownika), która wypisze na ekranie tekst „Hello World!”.

   a) Uruchom środowisko programistyczne NetBeans.

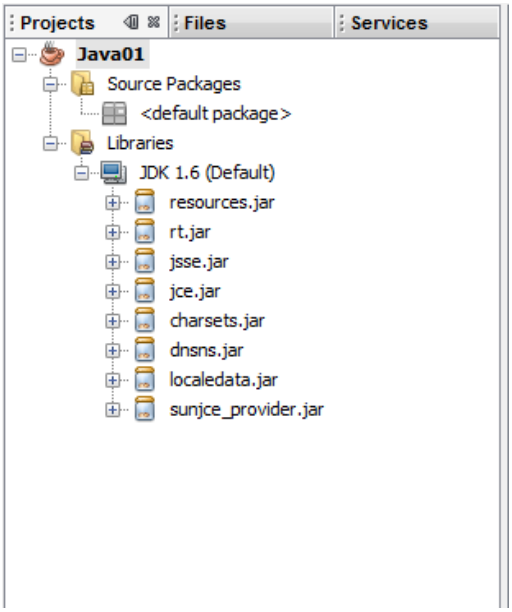
   b) Z menu File wybierz pozycję New Project. W oknie definiowania parametrów projektu, w panelu Categories, zaznacz pozycję Java, w panelu Projects zaznacz pozycję Java Application. Następnie naciśnij przycisk Next >.



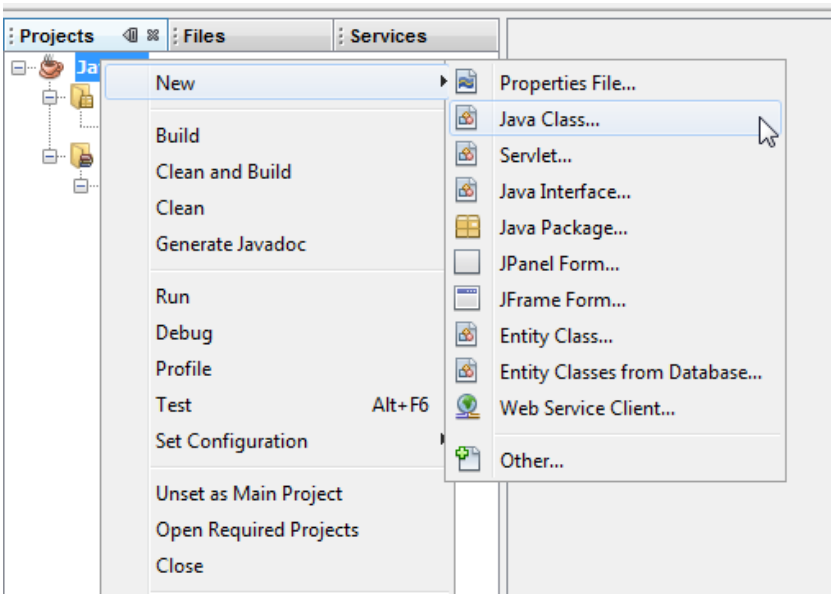

   c) W polu Project Name wpisz nazwę tworzonego projektu: **Java01.** Zlikwiduj zaznaczenie opcji Create Main Class, pozostaw zaznaczoną opcję Set as Main Project. Naciśnij przycisk Finish.



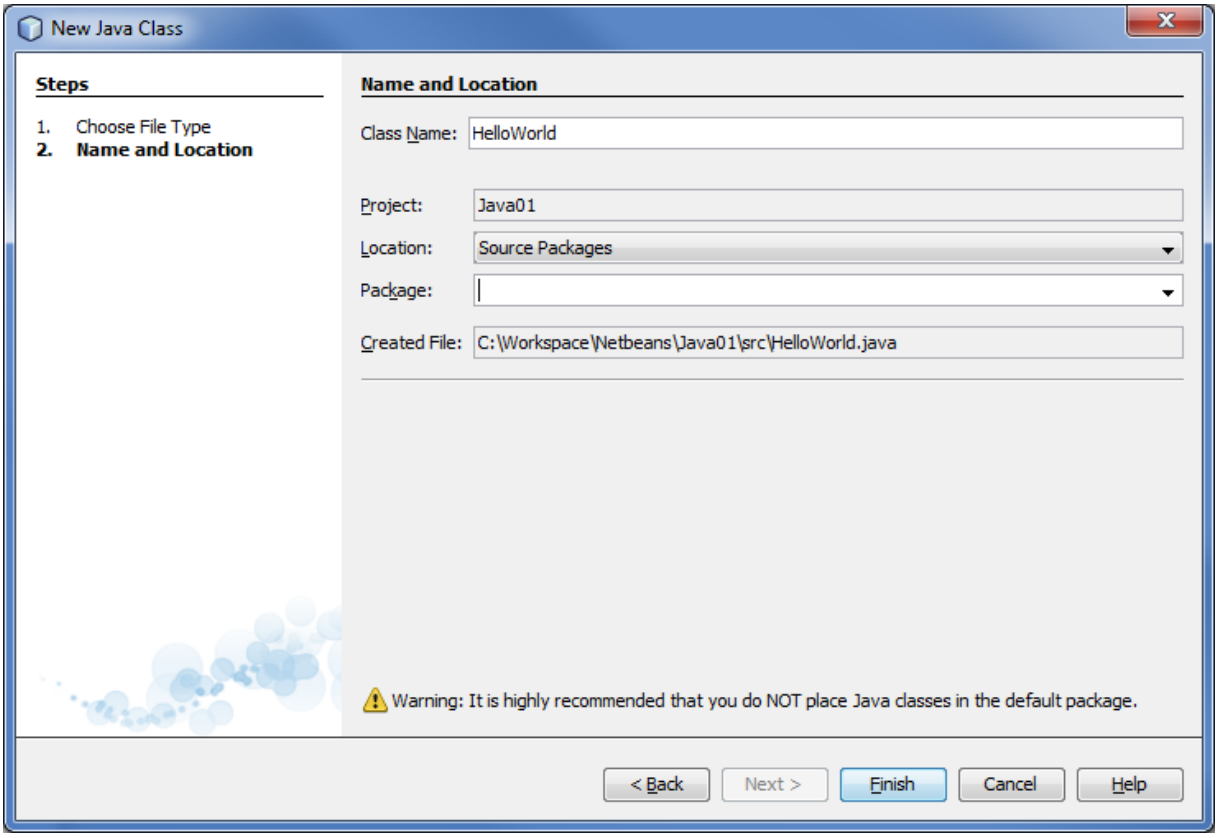

Struktura projektu powinna wyglądać tak, jak na poniższym rysunku. Zauważ, że projekt nie posiada na razie żadnych klas Javy (pusta zawartość gałęzi Source Packages).




d) Zdefiniujesz teraz klasę w projekcie. W tym celu kliknij prawym klawiszem myszy w obszar nazwy projektu i z menu kontekstowego wybierz pozycję New → Java Class.




e) Nazwa definiowanej klasy to `HelloWorld` (pole Class Name), klasa nie ma być umieszczona w żadnym pakiecie (pole Package pozostaw puste). Naciśnij przycisk Finish.



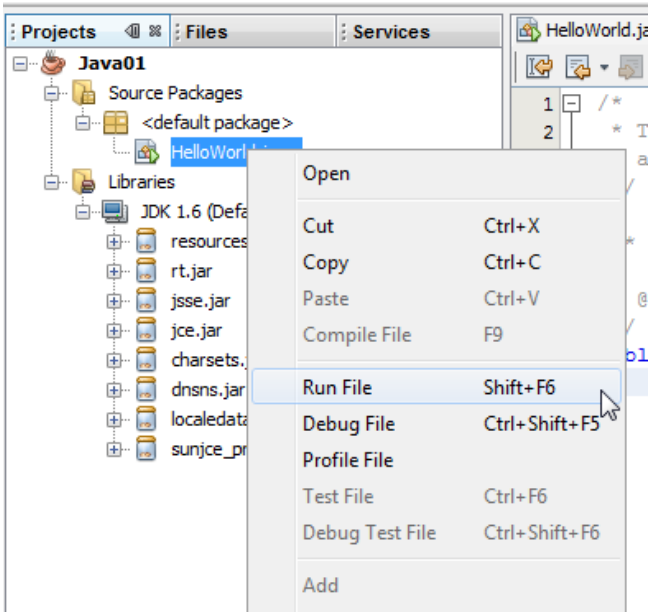
Kreator utworzył plik `HelloWorld.java` (nazwa pliku musi być zgodna z nazwą klasy, którą zawiera) z publiczną klasą `HelloWorld`. Klasa nie posiada na razie żadnych pól ani metod.

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *
 * @author Me
 */
public class HelloWorld {

}
```

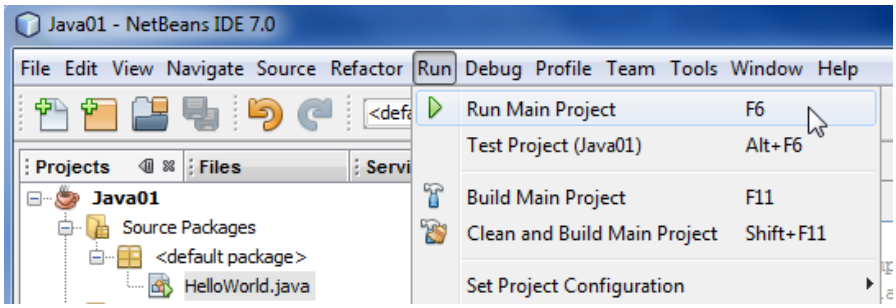
Klasa `HelloWorld` będzie zawiera tylko jedną, statyczną metodę `main`, od której będzie się rozpoczynało uruchomienie aplikacji. W metodzie `main` umieścisz kod, wypisujący na ekranie tekst „Hello World!”. Dodaj wewnątrz klasy `HelloWorld` poniższy kod.

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

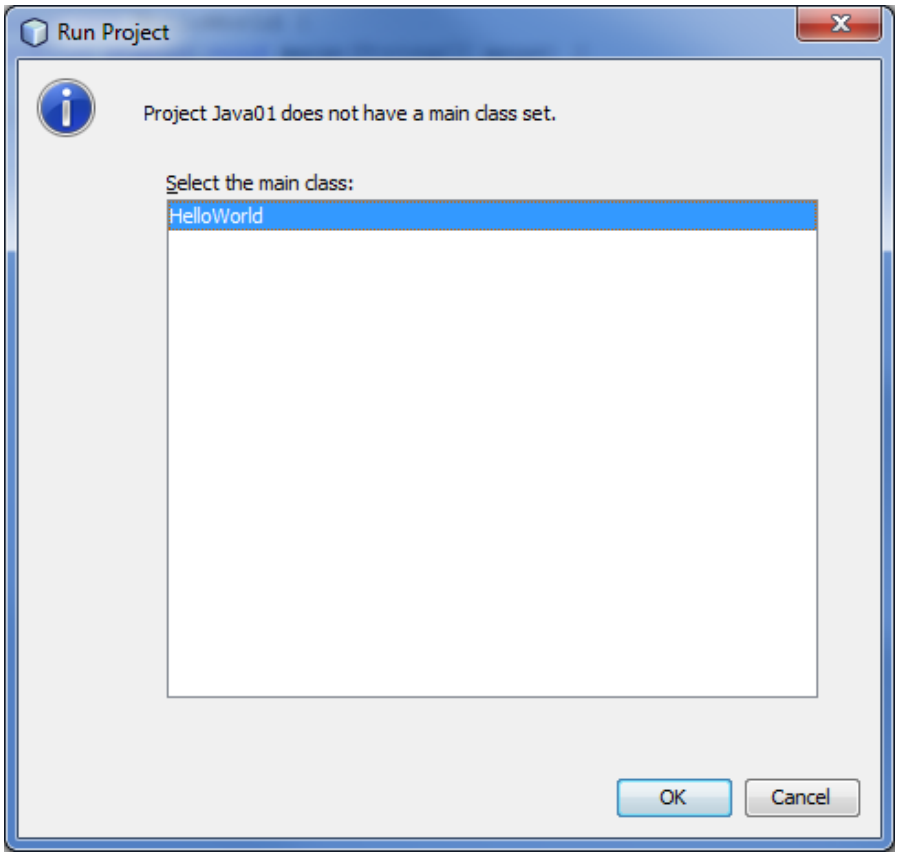
f) Spróbuj teraz uruchomić aplikację. Można to zrobić dwoma sposobami. Pierwszy to kliknięcie prawym klawiszem myszy na nazwę pliku z definicją klasy (`HelloWorld.java`) w oknie struktury projektu i wybranie pozycji Run File.



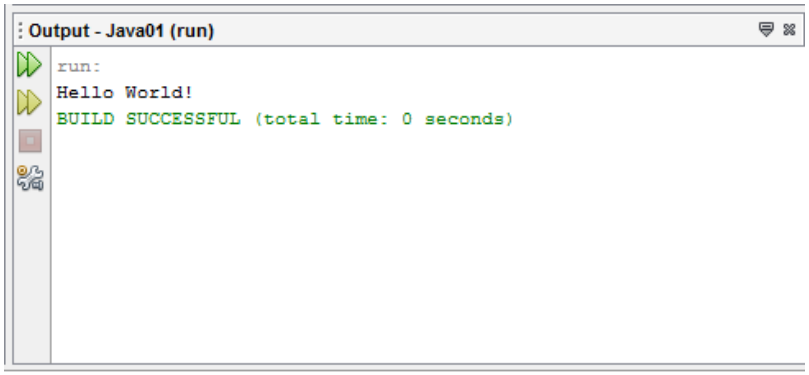
Drugi sposób to wybranie z głównego menu środowiska pozycji Run a następnie Run Main Project lub naciśnięcie przycisku , umieszczonego na pasku narzędziowym.




g) Tutaj trzeba dodatkowo wskazać główną klasę projektu (nie podaliśmy jej, definiując projekt). W tym celu zaznacz w oknie Run Project klasę `HelloWorld` i naciśnij przycisk OK.



Jeśli klasa została zdefiniowana poprawnie, w dolnej części środowiska, w oknie zatytułowanym Output – Java01 (run) powinien pojawić się wynik poprawnego wykonania programu – napis „Hello World!”.




Możesz teraz zamknąć projekt – kliknij prawym klawiszem myszy na nazwę projektu i z menu kontekstowego wybierz pozycję Close (projekt pozostanie zapisany na dysku, jednak przestanie być wyświetlany w oknie projektów; możesz do niego później wrócić, wczytując go przez wykonanie operacji File → Open Project).

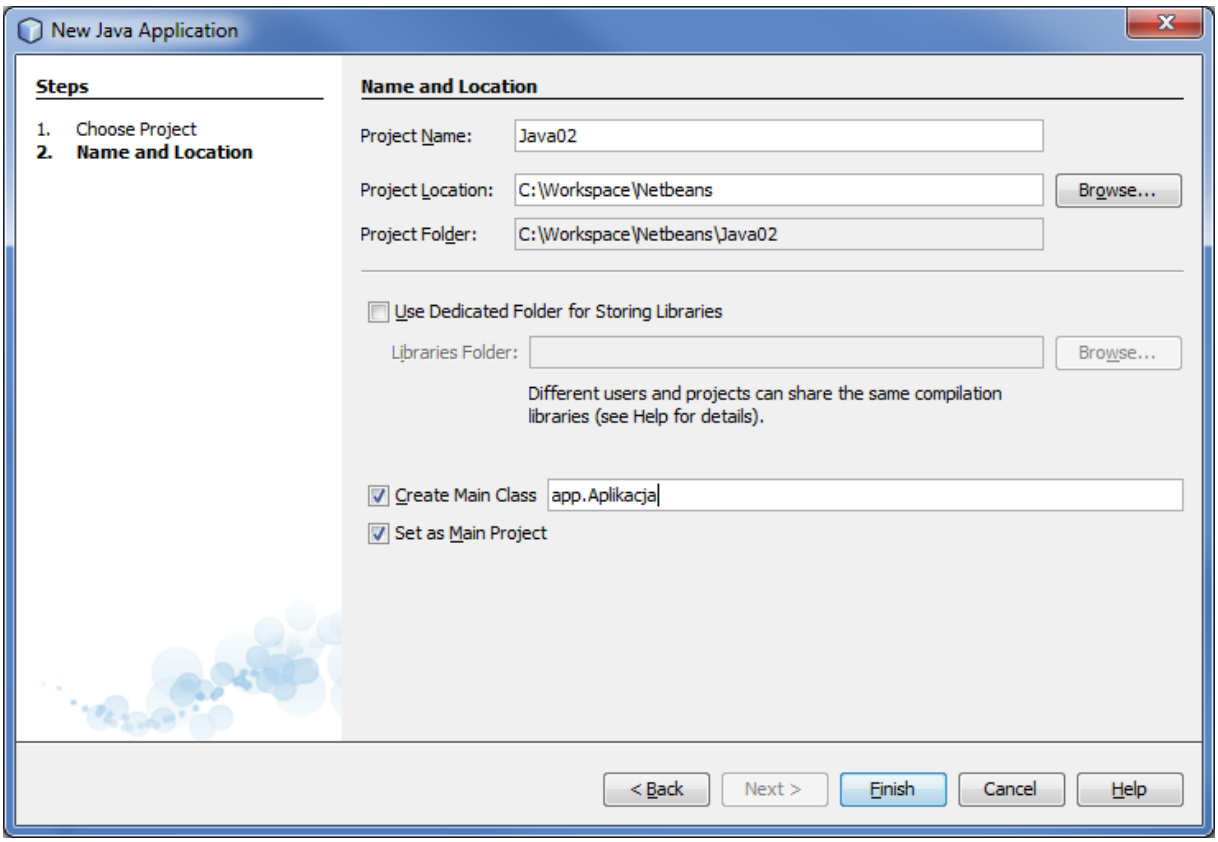
2. W drugim zadaniu zaimplementujesz kolejną aplikację konsolową, tym razem zawierającą dwie klasy: `Matematyka` w pakiecie `utils` i `Aplikacja` w pakiecie `app`:

   - `utils.Matematyka` będzie zawierała dwie publiczne metody statyczne: `silnia1` i `silnia2` przyjmujące jeden argument typu `long`, wyznaczające wartość silni podanej jako parametr liczby naturalnej. Pierwsza metoda powinna wyliczać silnię korzystając z instrukcji pętli, a druga korzystając z rekurencji. Na razie pomiń kwestię sprawdzania czy podany parametr jest nieujemny.
   - `app.Aplikacja` to klasa, od której będzie rozpoczynało się uruchomienie aplikacji (klasa musi zawierać metodę `main` o odpowiedniej sygnaturze). Metoda `main` powinna wyświetlić wartości silni liczb od 0 do 10 najpierw korzystając z pętli `while` i pierwszej implementacji silni, a następnie z pętli `for` i drugiej implementacji silni.

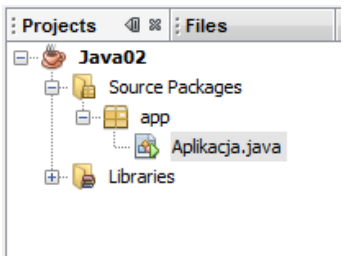
   Obie klasy mają zostać umieszczone w tym samym projekcie o nazwie **Java02**.

   a) Z menu File wybierz pozycję New Project. W oknie definiowania parametrów projektu, w panelu Categories, zaznacz pozycję Java, w panelu Projects zaznacz pozycję Java Application. Następnie naciśnij przycisk Next >.

   b) Nazwij projekt **Java02**, zaznacz opcję Create Main Class, w pole obok wpisz `app.Aplikacja` (to będzie nasza klasa projektu – od niej rozpocznie się wykonanie aplikacji), pole Set as Main Project pozostaw zaznaczone. Naciśnij przycisk Finish.

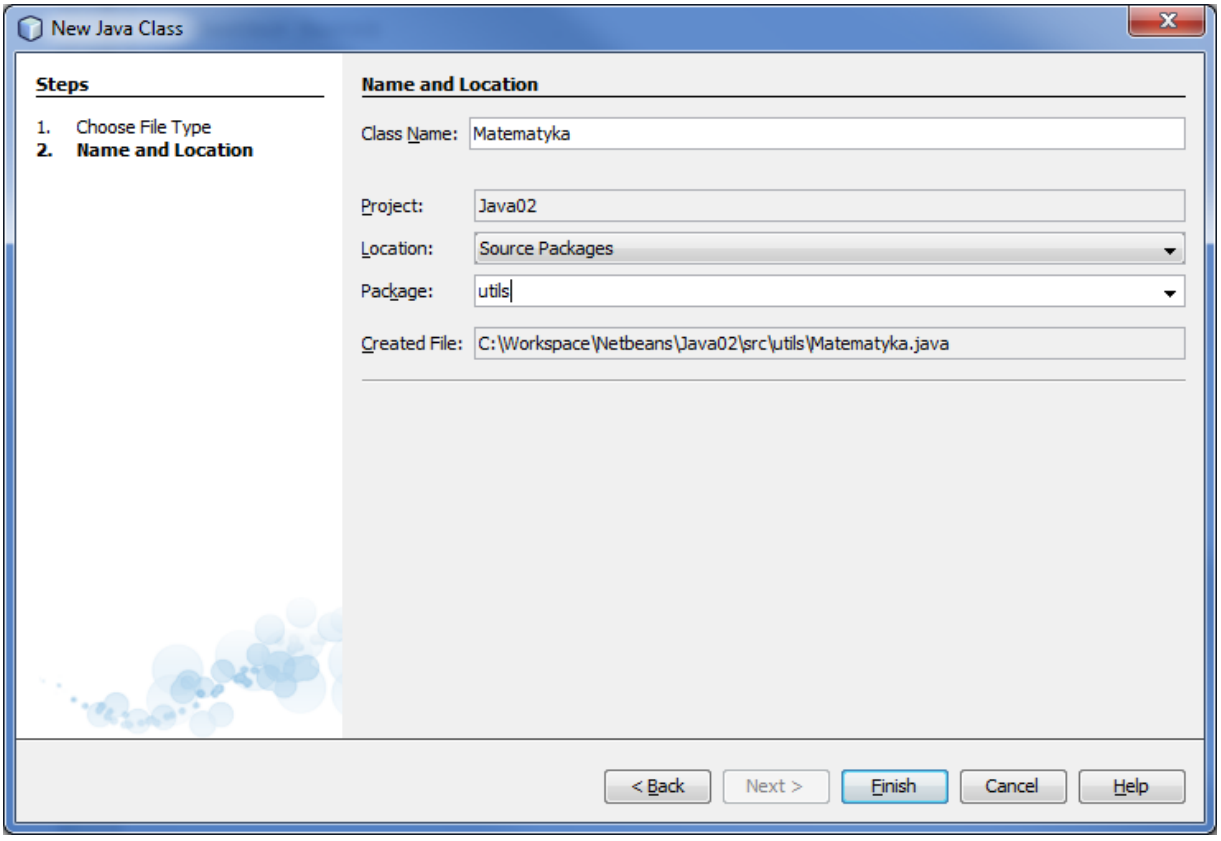


Zwróć uwagę na strukturę projektu – plik `Aplikacja.java` został umieszczony w gałęzi `app` – oznacza to, że klasa `Aplikacja` została umieszczona w pakiecie `app`.




Zanalizuj wygenerowany kod klasy `app.Aplikacja` – zwróć uwagę na komendę `package`. Zauważ, że klasa została wyposażona w statyczną funkcję `main` – to jest efekt zaznaczenia opcji Create Main Class przy definiowaniu projektu.

c) Zdefiniuj teraz drugą klasę, mianowicie `utils.Matematyka`. W tym celu kliknij prawym klawiszem myszy na nazwie projektu i z menu kontekstowego wybierz pozycję New → Java Class. Wpisz nazwę klasy (pole Class Name), nie zapomnij podać nazwy pakietu, w którym zostanie umieszczona klasa (pole Package). Naciśnij przycisk Finish.




W edytorze kodu powinno zostać otwarte nowe okno, wyświetlające kod nowo zdefiniowanej klasy. Zwróć uwagę na zmiany w oknie projektu.

d) Dodaj teraz do klasy `utils.Matematyka` publiczną, statyczną metodę `silnia1`, o jednym argumencie typu `long`, która wyliczy wartość silni korzystając z instrukcji pętli.

Spróbuj zdefiniować kod tej metody samodzielnie. Jeśli masz problemy, kod został zamieszczony w Dodatku 1.

e) Przejdź teraz do edycji klasy `app.Aplikacja`. Umieść w metodzie **`main`** wywołania metody silnia1 klasy `utils.Matematyka` dla wartości od 0 do 10 przy użyciu pętli `while`. Uzyskane wyniki mają zostać wypisane na ekranie w poniższej formie:

```
Test metody silnia1
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
```

Pamiętaj, że metoda `silnia1` znajduje się w innej klasie i pakiecie niż klasa `app.Aplikacja` – konieczne będzie zaimportowanie klasy `utils.Matematyka`. Kod klasy `app.Aplikacja` znajduje się w Dodatku 2. Uruchom aplikację.

f) Dodaj teraz do klasy `utils.Matematyka` drugą publiczną, statyczną metodę `silnia2`, o jednym argumencie typu long, która wyliczy wartość silni korzystając z rekurencji. Spróbuj zdefiniować kod tej metody samodzielnie. Jeśli masz problemy, kod został zamieszczony w Dodatku 3. Dodaj wywołania tej metody do metody `main` klasy `app.Aplikacja` przy użyciu pętli **`for`**, wynik ma być prezentowany w analogiczny sposób jak przy metodzie `silnia2`. Zmodyfikowaną postać klasy `app.Aplikacja` znajdziesz w Dodatku 4.

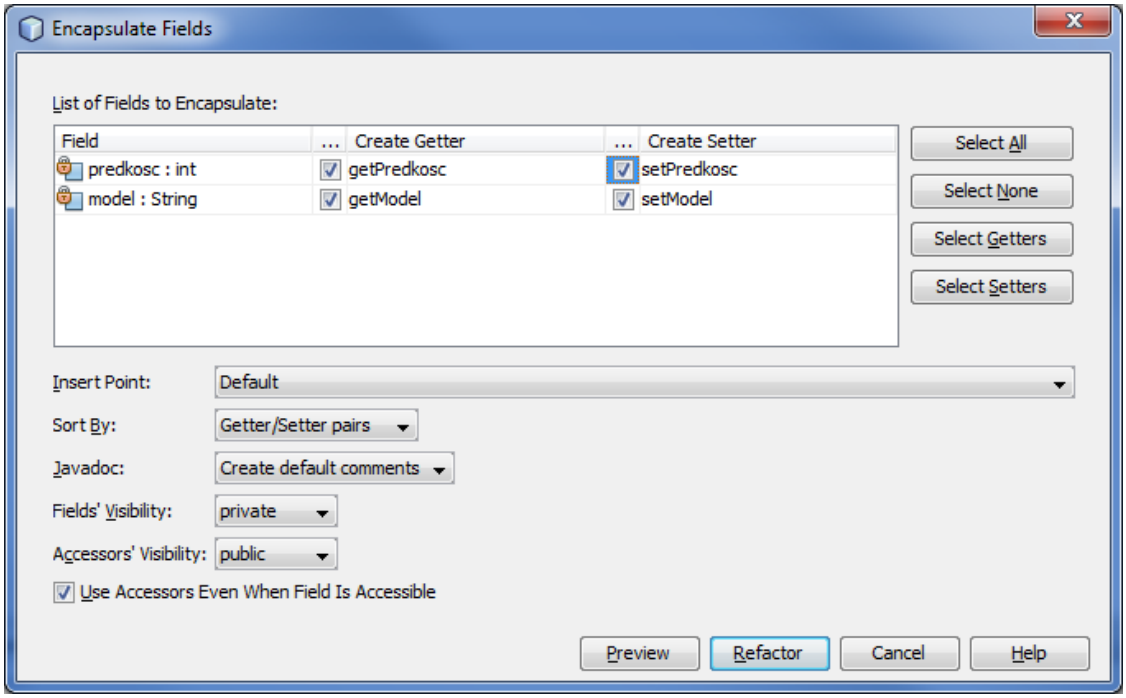
```
Test metody silnia2
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
```

g) Zmodyfikuj teraz obie metody, wyliczające silnię, w taki sposób, aby w sytuacji wywołania metod z ujemną wartością parametru rzucały wyjątek `java.lang.ArithmeticException`. Kod metod po modyfikacji możesz znaleźć w Dodatku 5. Przetestuj działanie metod – wywołaj obie z ujemną wartością parametru (np. -1). Czy zaobserwowałaś/eś komunikat o błędzie?

3. W trzecim zadaniu zaimplementujesz konsolową aplikację, wykorzystującą dziedziczenie i polimorfizm.

   a) Utwórz nowy projekt o nazwie **Java03**.

   b) Utwórz abstrakcyjną klasę `test.Pojazd` zawierającą:
   - Prywatne pola: `predkosc` typu `int` i `model` typu `String`
   - Publiczne metody `setPredkosc`, `setModel`, `getPredkosc`, `getModel` o odpowiednich argumentach i typach zwrotnych. Metody te możesz zdefiniować ręczenie lub wygenerować za pomocą kreatora. W tym celu:
     1. Kliknij prawym klawiszem myszy w obrębie edytora, z menu kontekstowego wybierz pozycję Refactor → Encapsulate Fields....
     2. W wyświetlonym dialogu zaznacz chęć generacji wszystkich metod naciskając przycisk Select All a następnie Refactor.




   - 2-argumentowy publiczny konstruktor ustawiający `predkosc` i `model` na podstawie wartości podanych jako parametry wywołania.
   - Publiczną metodę zwracającą `String` zawierający informacje o modelu pojazdu i jego prędkości np. w formacie: „model:XXX predkosc:YYY”. Nazwij metodę tak aby była wykorzystywana jako domyślna metoda zwracająca tekstową reprezentację obiektu (redefiniuj odpowiednią metodę odziedziczoną z `java.lang.Object`).

   c) Utwórz klasę `test.Samochod` dziedziczącą z `Pojazd`:
   - Zawierającą dodatkowo prywatne pole `liczbaDrzwi` i publiczne metody `getLiczbaDrzwi` i `setLiczbaDrzwi`.
   - 3-argumentowy konstruktor ustawiający wartości pól tworzonego obiektu.

   - Redefinicję publicznej metody zwracającej tekstową reprezentację obiektu, wywołującą metodę odziedziczoną i dodatkowo uwzględniającą liczbę drzwi.

d) Utwórz klasę `test.Samolot` dziedziczącą z `Pojazd`:
   - Zawierającą dodatkowo prywatne pole `liczbaMiejsc` i publiczne metody `getLiczbaMiejsc` i `setLiczbaMiejsc`.
   - 3-argumentowy konstruktor ustawiający wartości pól tworzonego obiektu.
   - Redefinicję publicznej metody zwracającej tekstową reprezentację obiektu, wywołującą metodę odziedziczoną i dodatkowo uwzględniającą liczbę miejsc.

e) Utwórz startową klasę `test.Test` i w jej metodzie `main` kolejno:
   - Utwórz tablicę do składowania 4 pojazdów.
   - Wstaw do tablicy 2 samochody i 2 samoloty
   - Przejdź tablicę pętlą `for` i wyświetl opisy pojazdów
   - Przejdź tablicę pętlą `foreach` i wyświetl opisy pojazdów.

f) Uruchom program z poziomu środowiska IDE.

g) Dodaj do aplikacji interfejs `test.Tuningowalny` zawierający metodę `zwiekszPredkosc` z parametrem typu `int`.

h) Spraw aby klasa `Samochod` implementowała ten interfejs. Dodaj w klasie `Samochod` implementację metody `zwiekszPredkosc`.

i) W metodzie `main` po wypełnieniu tablicy obiektami Pojazd zadeklaruj zmienną typu `Tuningowalny`. Przypisz do niej referencję do jednego z samochodów. Za pomocą zmiennej typu `Tuningowalny` zwiększ prędkość tego samochodu.

j) Uruchom aplikację.

4. Zadanie dla chętnych: Aplikacja operująca na tablicach liczb.

   a) Dodaj do aplikacji z punktu 2. klasę `utils.Tablice` zawierające trzy publiczne metody statyczne przyjmujące jeden argument będący tablicą wartości typu `int`: `maksimum` – zwracającą największą wartość z tablicy, `suma` – zwracającą sumę wartości z tablicy i `sortuj` – sortującą elementy tablicy malejąco, niezwracającą żadnej wartości.

   a) Dodaj w klasie `app.Aplikacja` kod sprawdzający działanie wszystkich metod klasy `utils.Tablice`.

   b) Uruchom aplikację i przetestuj jej działanie.

**Dodatek 1.** Kod metody `silnia1` klasy `utils.Matematyka`.

```
    public static long silnia1(long n) {
        long silnia = 1;
        for (int i = 1; i <= n; i++) {
            silnia = silnia * i;
        }
        return silnia;
    }
```

**Dodatek 2.** Kod metody `main` klasy `app.Aplikacja`.

```
package app;

import utils.Matematyka;

public class Aplikacja {

    public static void main(String[] args) {

        System.out.println("Test metody silnia1");
        int i = 0;
        while(i <= 10) {
            System.out.println(i + "! = " + Matematyka.silnia1(i++));
        }
}
```

**Dodatek 3.** Kod metody `silnia2` klasy `utils.Matematyka`.

```
    public static long silnia2(long n) {
        if (n == 0) {
            return 1;
        } else {
            return n * silnia2(n - 1);
        }
    }
```

**Dodatek 4.** Kod metody `main` klasy `app.Aplikacja`.

```
package app;

import utils.Matematyka;

public class Aplikacja {

    public static void main(String[] args) {

        System.out.println("Test metody silnia1");
        int i = 0;
        while(i <= 10) {
            System.out.println(i + "! = " + Matematyka.silnia1(i++));
        }

        System.out.println("Test metody silnia2");
        for (int j = 0; j <= 10; j++) {
            System.out.println(j + "! = " + Matematyka.silnia2(j));
        }
    }
}
```

**Dodatek 5.** Metody `silnia1` i `silnia2` wyposażone w detekcję wywołania z ujemną wartością parametru.

```
public static long silnia1(long n) throws java.lang.ArithmeticException {
        if (n < 0) {
            throw new java.lang.ArithmeticException();
        }

        long silnia = 1;

        for (int i = 1; i <= n; i++) {
            silnia = silnia * i;
        }
        return silnia;
    }

    public static long silnia2(long n) throws java.lang.ArithmeticException {
        if (n < 0) {
            throw new java.lang.ArithmeticException();
        }

        if (n == 0) {
            return 1;
        } else {
            return n * silnia2(n - 1);
        }
    }
```