Podstawy języka JAVA

Wykład nr 5 – Pola i metody statyczne. Parametry metod. Dzienniki w Java.

Wyższa Szkoła Informatyki Stosowanej i Zarządzania
pod auspicjami Polskiej Akademii Nauk

Java – pola statyczne

- Na wcześniejszych wykładach przedstawiono przykłady wykorzystania funkcji main. Zawsze opatrzona była ona modyfikatorem static.
- W danej klasie może być tylko jeden egzemplarz pola określanego jako statyczne.
- W przeciwieństwie do pól statycznych, każdy obiekt klasy ma własną kopię pól, które nie są statyczne.
- Przykładem wykorzystania pola statycznego jest nadawanie unikalnych numerów id dla każdego nowo tworzonego pracownika.

Java – pola statyczne

```
public class Employee {

    private static int globalID=1;
    public int ID;
    private String name;
    private String surname;
    public Employee(String name, String surname)
    {
        ID=globalID;
        globalID++;
        this.name=name;
        this.surname=surname;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Employee e=null;
        for(int i=0;i<100;i++)
        {
            e = new Employee("a","b");
        }
        System.out.println(Employee.globalID);
    }
}
```

Java – pola statyczne

- Każdy obiekt klasy Employee będzie miał własne pole ID.
- Pole globalID jest współdzielone przez wszystkie obiekty klasy Employee.
- Przykładowo jeżeli zostanie utworzonych 1000 obiektów klasy Employee to powstanie 1000 składowych obiektu o nazwie ID, ale pole globalID będzie tylko jedno.
- Pole globalID będzie istniało jeżeli nawet nie zostanie utworzony żaden obiekt klasy Employee.
- Pole globalID należy do klasy, a nie konkretnego obiektu.

Java – stałe statyczne

- Zmienne statyczne wykorzystuje się w małej ilości przypadków.
- Dużo częściej wykorzystuje się stałe statyczne.
- Przykładowo w klasie Math znajduje się stała statyczna PI, do której dostęp można uzyskać poprzez Math.PI.
- Stała statyczna określana jest słowami kluczowymi static final.

```
public class Math
{
    . . .
    public static final double PI = 3.14159265358979323846;
    . . .
}
```

Java – stałe statyczne

- Gdyby słowo kluczowe static zostało pominięte to PI byłoby zwykłym polem klasy Math.
- W związku z tym dostęp do tego pola odbywałby się poprzez obiekt klasy Math.
- Dodatkowo każdy obiekt tej klasy miałby składową PI.
- Innym przykładem często używanej stałej statycznej jest System.out.
- Jej deklaracja w klasie System wygląda następująco:

```
public class System
{
    . . .
    public static final PrintStream out = . . .;
    . . .
}
```

Java – metody statyczne

- Metody statyczne nie działają na obiektach!
- Przykładem metody statycznej jest metoda pow z klasy Math.
- Wywołanie Math.pow(x,a) powoduje podniesienie x do potęgi a.
- Do wywołania tej metody nie jest potrzebny żaden obiekt klasy Math.
- Innymi słowy, nie ma parametru niejawnego (obiektu, na rzecz którego wywoływana jest metoda).
- Metody statyczne najprościej zapamiętać jako metody, które nie mają parametru this.

Java – metody statyczne

- Przykładowo metoda statyczna w klasie Employee nie ma dostępu do zmiennej ID, ponieważ nie działa ona na obiektach.
- Metoda statyczna ma natomiast dostęp do pól statycznych w klasie Employee.

```
public class Employee {

    public static int globalID=1;
    public static int getGlobalID()
    {
        return globalID;
    }

    public Employee()
    {

    }
}
```

```
public class Main {
    public static void main(String[] args) {
        int id = Employee.getGlobalID();
        System.out.println(Employee.getGlobalID());
    }
}
```

Java – metody statyczne

- Metody statyczne mają dwojakie zastosowanie:
  - Kiedy metoda nie wymaga dostępu do stanu obiektu, ponieważ wszystkie wymagane jej parametry są dostarczane w postaci parametrów jawnych np. Math.PI
  - Kiedy metoda wymaga jedynie dostępu do pól statycznych swojej klasy (np. zwiększanie ID pracowników).

Java – metoda main

- Metody statyczne można wywoływać nie mając utworzonego żadnego obiektu klasy.
- W ten sposób np. wywołuje się Math.pow.
- Właśnie z tego powodu, że nie jest potrzebny obiekt do wywołania metody statycznej, metoda main jest statyczna.
- Metoda main nie działa na żadnym obiekcie.
- Przy uruchamianiu programu, nie ma w nim powołanych żadnych obiektów.
- To właśnie metoda main odpowiedzialna jest za tworzenie i uruchamianie obiektów wymaganych przez program.

Java – metoda main

- Każda klasa może posiadać własną metodę main.
- Jest to bardzo użyteczny mechanizm do przeprowadzania testów jednostkowych.
- W celu uruchomienia metody main w klasie Employee należy wykonać polecenie java Employee. Jeżeli klasa Employee jest częścią większej aplikacji i zostanie uruchomione polecenie java Aplikacja to metoda main w klasie Employee zostanie pominięta.

Java – parametry metod

- Istnieją różne sposoby przekazywania parametrów do metod (lub funkcji) w różnych językach programowania.
- Termin wywołanie przez wartość (ang. call by value) oznacza, że metoda odbiera tylko wartość dostarczoną przez wywołującego.
- Termin wywołanie przez referencję (ang. call by reference) oznacza, że metoda odbiera lokalizację zmiennej dostarczonej przez wywołującego.
- W związku z tym metoda może zmodyfikować wartość zmiennej przekazanej przez referencję, ale nie może tego zrobić ze zmienną przekazaną przez wartość.

Java – parametry metod

- Termin wywołanie przez nazwę (ang. call by name) ma już tylko znaczenie historyczne, ponieważ żaden ze współczesnych języków programowania nie wykorzystuje tego sposobu przekazywania parametrów do metod.
- W Java zawsze stosowane są wywołania przez wartość.
- Oznacza to, że metoda otrzymuje kopię wartości wszystkich parametrów, a więc nie może zmodyfikować wartości przekazanych do niej zmiennych.

Java – parametry metod

- W przedstawionym przykładzie, wiadomo że po wykonaniu metody raiseSalaray, wartość zmiennej percent będzie wynosiła 10.

```
public class Main {
    public static void main(String[] args) {
        Employee e=new Employee("Adam","Adamiak");
        int percent=10;
        e.RaiseSalary(percent);
    }
}
```

- Niezależnie od implementacji metody RaiseSalary w klasie Employee programista ma pewność że wartość percent po wykonaniu metody będzie wynosiła 10.

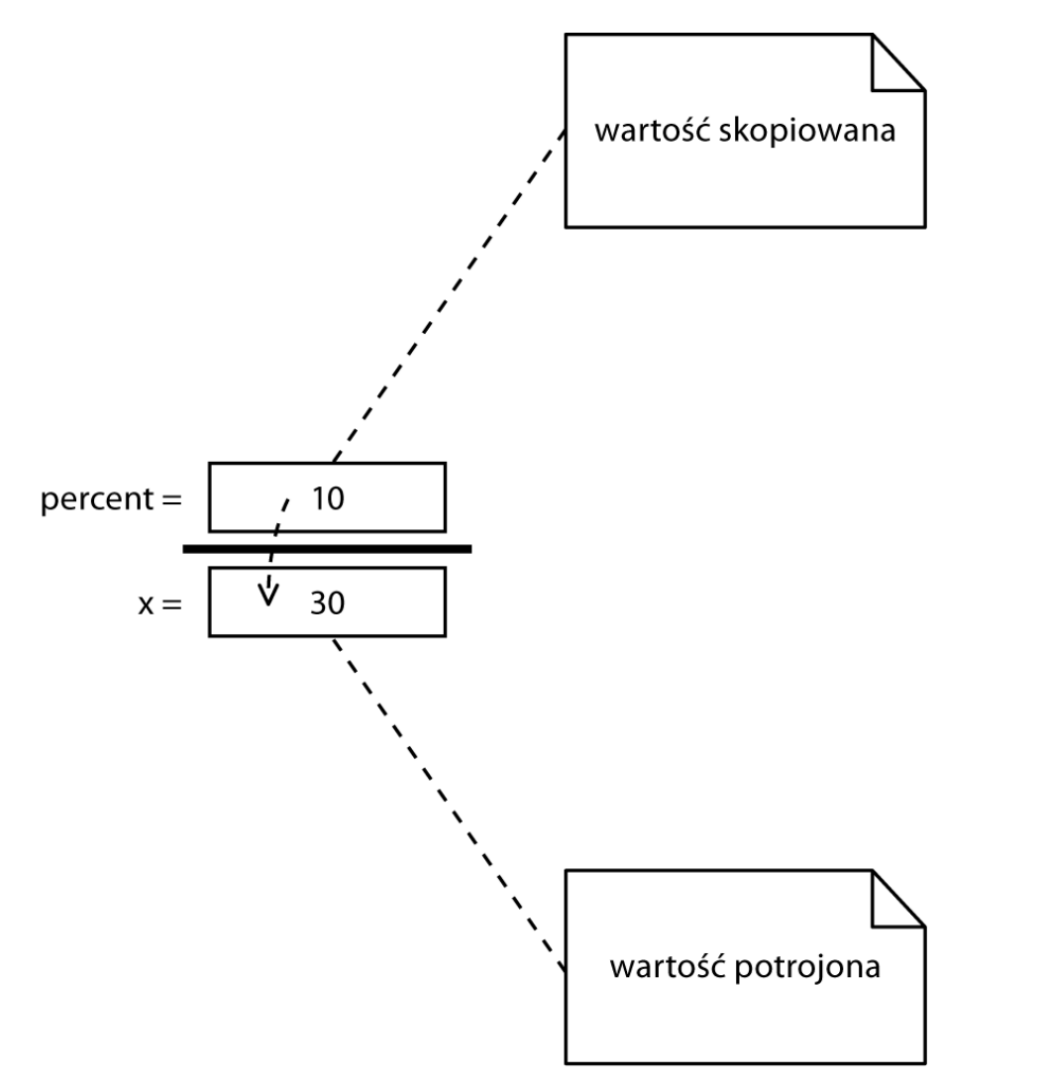
Java – parametry metod

- Metoda RaiseSalary próbuje potroić wartość przekazanego parametru, jednak zmienna percent po jej wykonaniu dalej ma wartość 10.
- Kroki wykonania przedstawionego fragmentu kodu są następujące:
  - Parametr x inicjowany jest kopią wartości zmiennej percent.
  - Wartośc zmiennej x jest potrojona i wynosi 30, ale zmienna percent dalej ma wartość 10.
  - Metoda kończy działanie, a zmienna x nie jest używana.

```
public static void RaiseSalary(int x)
{
    x = 3*x;
}
```

```
public class Main {
    public static void main(String[] args) {
        int percent=10;
        Employee.RaiseSalary(percent);
    }
}
```

Java – parametry metod



Java – parametry metod

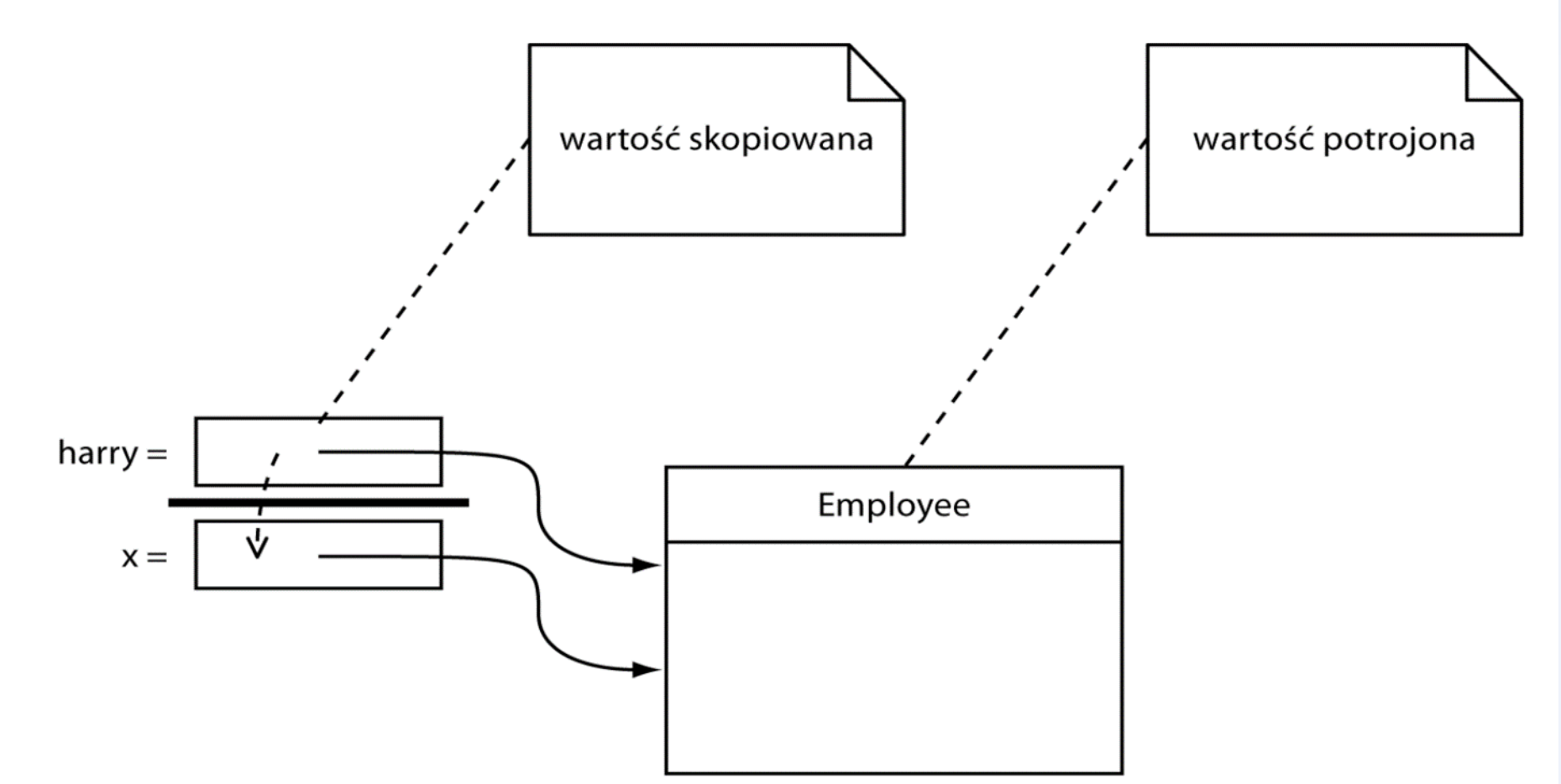
- W Java są jednak dwa rodzaje parametrów metod:
  - Typy podstawowe,
  - Referencje do obiektów.
- Z poprzedniego slajdu wiadomo już, że nie można zmienić wartości parametru typu podstawowego.
- Inaczej wygląda sytuacja w przypadku parametrów typu obiektowego.
- Można w bardzo prosty sposób napisać metodę, która potraja pensję pracownika.

```
public static void main(String[] args) {
    Employee harry = new Employee("Adam","Adamski");
    System.out.println(harry.getSalaray());
    Employee.RaiseSalary(harry);
    System.out.println(harry.getSalaray());
}
```

```
public static void RaiseSalary(Employee x)
{
    x.salary=3*x.salary;
}
```

- W przedstawionym przykładzie kolejność wykonywania jest następująca:
  - Zmienna x inicjowana jest kopią wartości obiektu harry tzn. referencją do obiektu.
  - Metoda RaiseSalary wywoływana jest na rzecz tej referencji. Pensja obiektu klasy Employee, do którego odwołują się zmienna harry oraz zmienna x jest zwiększana.
  - Metoda kończy działanie i zmienna x nie jest już używana. Zmienna harry natomiast dalej odwołuje się do obiektu, którego pensja została zwiększona.

Java – parametry metod



Java – parametry metod

- Napisanie metody, która zmienia stan parametru w postaci obiektu jest czynnością prostą i bardzo często wykorzystywaną w Java.
- Prostota wynika z tego, że metoda odbiera kopię referencji do obiektu. Zarówno oryginalny obiekt jak i kopia refrencji odwołują się do tego samego obiektu.
- W wielu językach programowania metody do metod można przekazywać zarówno przez wartość jak i referencję.
- Niektórzy programiści twierdzą, że w Java dla obiektów stosowane jest wywołanie przez referencję.
- Jest to nieprawda.

Java – parametry metod

- W klasie Employee napisano metodę, która zamienia ze sobą dwa obiekty.

```
public class Employee {

    private String name;

    public Employee (String name)
    {
        this.name=name;
    }

    public static void swap (Employee x, Employee y)
    {
        Employee tmp=x;
        x=y;
        y=tmp;
        System.out.println(x.name);
        System.out.println(y.name);
    }

    public String getName()
    {
        return this.name;
    }
}
```

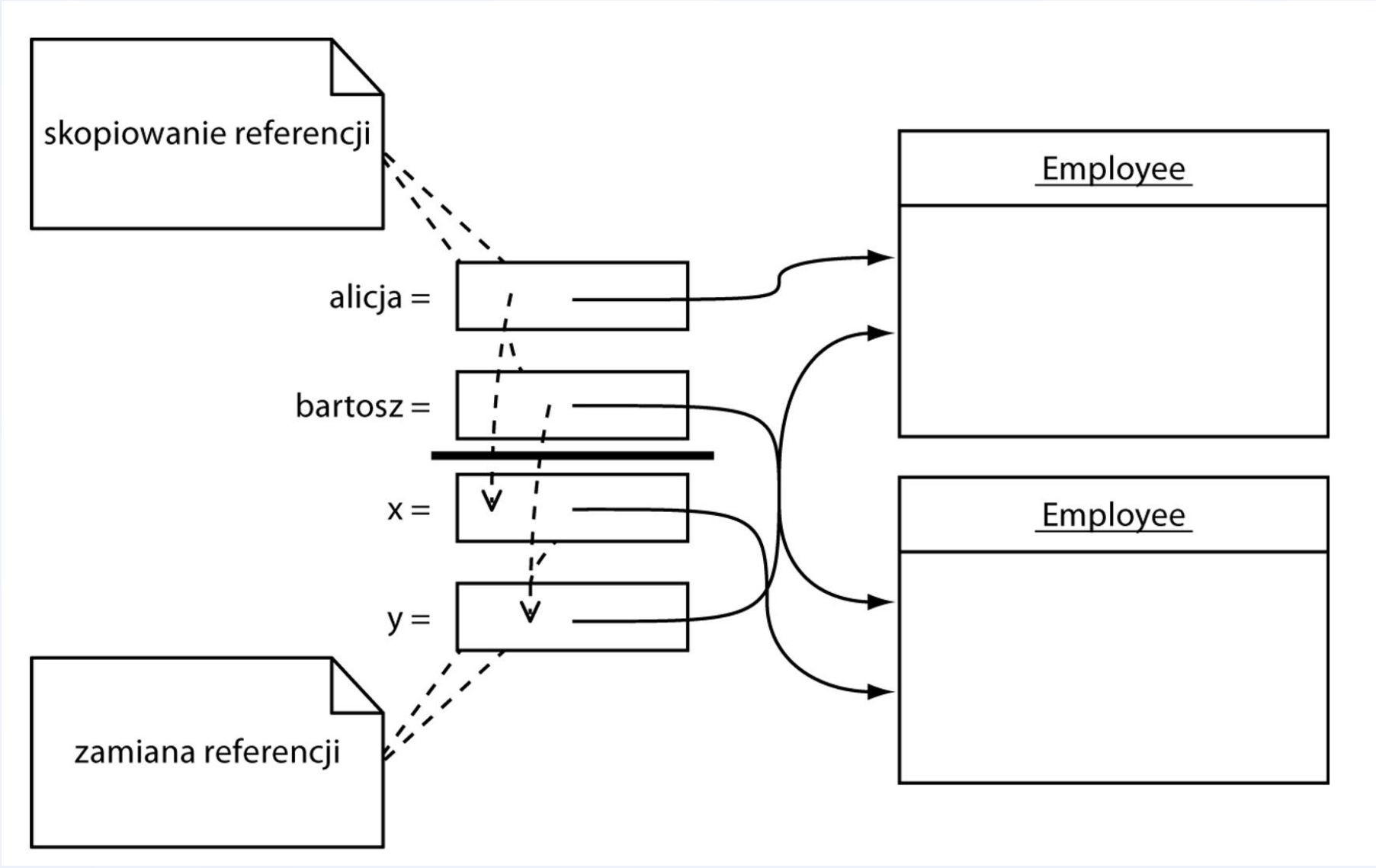
```
public class Main {
    public static void main(String[] args) {
        Employee a = new Employee("alicja");
        Employee b = new Employee("bartosz");
        Employee.swap(a, b);
        System.out.println(a.getName());
        System.out.println(b.getName());
    }
}
```

```
bartosz
alicja
alicja
bartosz
```

Java – parametry metod

- Gdyby w Java stosowane były wywołania przez referencję to obiekty a i b zostałyby zamienione miejscami.
- Przedstawiona metoda swap nie zmienia jednak referencji do obiektów przechowywanych w zmiennych a i b.
- Parametry x i y metody swap są inicjalizowane kopiami referencji.
- Po zakończeniu działania metody zmienne x i y wychodzą z użycia,.
- Oryginalne zmienne a i b nadal odwołują się do tych samych obiektów, do których odwoływały się przed wywołaniem metody swap.

Java – parametry metod



Java – parametry metod

- Zasady dotyczące tego co można, a czego nie można zrobić z parametrami metod w Java:
  - Metoda nie może zmodyfikować parametru typu podstawowego.
  - Metoda może zmienić stan obiektu przekazanego jako parametr.
  - Metoda nie może sprawić, aby parametr obiektowy zaczął odwoływać się do innego obiektu.

Java – dzienniki

- Programiści bardzo często wypisują na ekran zmienne lub komunikaty w celu sprawdzania poprawności działania.
- Po zdiagnozowaniu problemu taka linia jest zazwyczaj komentowana, bądź usuwana z programu.
- API (ang. Application Programming Interface) o nazwie Logging pozwala uniknąć wyżej opisanych sytuacji.
- Zalety API Logging:
  - Możliwość łatwego włączenia/wyłączenia wybranego poziomu bądź wszystkich rekordów w dzienniku.
  - Rejestracja danych jest bardzo mało kosztowna, co oznacza że jej pozostawienie w aplikacji powoduje minimalne opóźnienia.

Java – dzienniki

- Rekordy dziennika można wysyłać do różnych procedur obsługi, które np. wypiszą je na ekran lub zapiszą do pliku.
- Zarówno rejestratory (ang. logger) jak i procedury obsługi dziennika umożliwiają filtrowanie rekordów. Filtr pozwala na odsiewanie niepotrzebny zapisów za pomocą kryteriów podanych przez programistę.
- Rekordy w dzienniku mogą być formatowane w różny sposób np. zwykły tekst lub XML.
- Aplikacja może posiadać wiele rejestratorów, ułożonych w hierarchicznej strukturze.

Java – dzienniki

- Podstawową operacją, która pozwala na zapis do dziennika jest wywołanie metody na obiekcie globalnego rejestratora.

```
import java.util.logging.Level;
import java.util.logging.Logger;

public class Main {
    public static void main(String[] args) {
        Logger.getGlobal().setLevel(Level.INFO);
        Logger.global.info("To jest wiadomość dziennika");
    }
}
```

```
gru 07, 2018 11:41:02 PM Main main
INFO: To jest wiadomość dziennika
```

- Przed pierwszym logowaniem wymagane jest ustawienie poziomu logowania dziennika za pomocą metody setLevel.
- Wyłączenie logowania do dziennika uzyskuje się poprzez następujący zapis w kodzie programu:

```
Logger.getGlobal().setLevel(Level.OFF);
```

Java – dzienniki

- W najnowszej wersji Java logowanie odbywa się za pomocą:

```
public class Main {
    public static void main(String[] args) {
        Logger.getGlobal().log(Level.INFO, "To jest wiadomość");
    }
}
```

- Wykorzystanie globalnego logger w całej aplikacji nie jest jednak dobrym podejściem.
- Bardzo często konieczne jest logowanie informacji o różnych poziomach w ramach różnych klas.
- W tym celu należy powołać własne obiekty loggerów.

Java – dzienniki

- W najnowszej wersji Java logowanie odbywa się za pomocą:

```
public class Main {
    private static final Logger MainLogger = Logger.getLogger("MainLogger");
    public static void main(String[] args) {
        MainLogger.log(Level.INFO, "To jest wiadomość MainLogger");
    }
}
```

```
gru 07, 2018 11:53:35 PM Main main
INFO: To jest wiadomość MainLogger
```

- Loggery mogą tworzyć hierarchię dziedziczenia np. MainLogger -> MainLogger.potomek.
- Potomne loggery dziedziczą np. poziom logowania.

Java – dzienniki

- Istnieje siedem poziomów ważności komunikatów:
  - SEVERE,
  - WARNING,
  - INFO,
  - CONFIG,
  - FINE,
  - FINER,
  - FINEST.
- Domyślnie dla loggerów są właczone trzy pierwsze poziomy.
- W celu wykorzystania pozostałych należy wywołać metodę setLevel z parametrem, który odpowiada wybranemu poziomowi komunikatów.

Java – dzienniki

- Następujące wywołanie:

```
MainLogger.setLevel(Level.FINE);
```

Spowoduje, że logowane będą wszystkie komunikaty od poziomu FINE w górę.

- W celu włączenia wszystkich poziomów należy wykorzystać Level.ALL
- W celu wyłączenia wszystkich poziomów należy wykorzystać Level.OFF
- Każdy z przedstawionych poziomów posiada swoją własną metodę do logowania.
- Można również wywołać metode log, która przyjmuje poziom logowanego komunikatu.
- Loggery mogą tworzyć hierarchię dziedziczenia np. MainLogger -> MainLogger.potomek.
- Potomne loggery dziedziczą np. poziom logowania.

Java – dzienniki

```
public class Main {
    private static final Logger MainLogger = Logger.getLogger("MainLogger");
    public static void main(String[] args) {
        MainLogger.setLevel(Level.FINE);
        MainLogger.severe("To jest poziom Severe");
        MainLogger.info("To jest poziom INFO");
        MainLogger.finest("To jest poziom FINEST");
    }
}
```

```
gru 08, 2018 12:01:36 AM Main main
SEVERE: To jest poziom Severe
gru 08, 2018 12:01:36 AM Main main
INFO: To jest poziom INFO
```

Java – dzienniki

```
public class Main {
    private static final Logger MainLogger = Logger.getLogger("MainLogger");
    public static void main(String[] args) {
        MainLogger.setLevel(Level.ALL);
        MainLogger.log(Level.SEVERE,"To jest poziom Severe");
        MainLogger.log(Level.INFO,"To jest poziom INFO");
    }
}
```

```
gru 08, 2018 12:07:01 AM Main main
SEVERE: To jest poziom Severe
gru 08, 2018 12:07:02 AM Main main
INFO: To jest poziom INFO
```

Java – dzienniki

- W domyślnym rekordzie dziennika znajduje się nazwa klasy i metody wywołującej logowanie zdarzenia. Informacje te są pobierane ze stosu wywołań.
- Maszyna wirtualna Java może jednak zoptymalizować wykonanie aplikacji i szczegółowe informacje nie będą dostępne.
- W celu przekazania szczegółowych informacji o metodzie oraz klasie wywołującej logowanie można wykorzystać metodę logp.

```
public class Main {
    private static final Logger MainLogger = Logger.getLogger("MainLogger");
    public static void main(String[] args) {
        MainLogger.setLevel(Level.ALL);
        MainLogger.logp(Level.INFO, "Main Class", "Metoda Main", "Przykład logp");
    }
}
```

```
gru 08, 2018 12:12:29 AM Main Class Metoda Main
INFO: Przykład logp
```

Java – dzienniki

- Zapis do dziennika jest często wykorzystywany w celu rejestrowania wyjątków.
- Są dwie metody, które zapisują w dziennika opis wyjątku: throwing oraz log.

```
public class Main {
    private static final Logger MainLogger = Logger.getLogger("MainLogger");
    public static void main(String[] args) {
        MainLogger.setLevel(Level.ALL);
        IOException exception = new IOException("...");
        MainLogger.throwing("MainClass", "Metoda Main", exception);

    }
}
```

- Metoda throwing zapisuje informację na poziomie FINER i komunikat zaczyna się od słowa THROW.

Java – dzienniki

- Domyślnie rejestratory wysyłają rekordy do obiektu typu ConsoleHandler, który drukuje przesłanie wiadomości w strumieniu System.err.
- Rejestrator wysyła rekord do nadrzędnego obiektu Handler, a przodek najwyższego poziomu (o nazwie „”) ma obiekt typu ConsoleHandler.
- Podobnie jak rejestratory, obiekty Handler mają poziomy rejestracji.
- W celu zapisania rekordu zarówno poziom rejestratora jak i obiektu Handler musi być ustawiony na odpowiedni poziom.
- Domyślny poziom domyślnego obiektu Handler ustawiony jest na INFO.

Java – dzienniki

- W związku z tym nie jest możliwe logowanie zdarzeń np. na poziomie FINE.
- W celu zmiany poziomu obiektu typu Handler można dokonać zmianę w pliku konfiguracyjnym Java, bądź samemu powołać obiekt typu Handler.

```
import java.util.logging.ConsoleHandler;
import java.util.logging.Handler;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Main {
    private static final Logger MainLogger = Logger.getLogger("MainLogger");
    public static void main(String[] args) {
        MainLogger.setLevel(Level.FINEST);
        MainLogger.setUseParentHandlers(false);
        Handler handler = new ConsoleHandler();
        handler.setLevel(Level.FINEST);
        MainLogger.addHandler(handler);
        MainLogger.finest("To jest poziom FINEST");
    }
}
```

```
gru 08, 2018 12:27:34 AM Main main
FINEST: To jest poziom FINEST
```

Java – dzienniki

- Domyślnie rejestrator wysyła rekordy zarówno do swoich własnych obiektów Handler, jak i obiektów Handler swojego przodka.
- Utworzony rejestrator jest potomkiem pierwotnego rejestratora, który wysyła wszystkie rekordy poziomu INFO lub wyższego do konsoli.
- W celu uniknięcia duplikatów komunikatów od poziomu INFO w zwyż, należy wywołać metodę useParentHandler z parametrem false.

Java – dzienniki

```
public class Main {
    private static final Logger MainLogger = Logger.getLogger("MainLogger");
    public static void main(String[] args) {
        MainLogger.setLevel(Level.FINEST);
        MainLogger.setUseParentHandlers(true);
        Handler handler = new ConsoleHandler();
        handler.setLevel(Level.FINEST);
        MainLogger.addHandler(handler);
        MainLogger.info("To jest poziom INFO");
    }
}
```

```
gru 08, 2018 12:31:37 AM Main main
INFO: To jest poziom INFO
gru 08, 2018 12:31:37 AM Main main
INFO: To jest poziom INFO
```

Java – dzienniki

- W celu wysyłania rekordów w inne miejsce niż konsola należy utworzyć inne obiekty typu Handler.
- API dzienników udostępnia dwa przydatne obiekty typu Handler: FilerHandler oraz SocketHandler.
- Obiekt typu Sockethandler wysyła logowane komunikaty do określonego hosta i na określony port.
- Obiekt typu FileHandler zapisuje rekordy do pliku.
- Plik zapisywany jest w lokalizacji użytkownika w formacie XML.
- Istnieje również możliwość podania ścieżki, gdzie plik dziennika ma zostać utworzony.

Java – dzienniki

```java
public class Main {
    private static final Logger MainLogger = Logger.getLogger("MainLogger");
    public static void main(String[] args) throws SecurityException, IOException {
        MainLogger.setLevel(Level.INFO);
        FileHandler file_handler = new FileHandler("c:\\java\\javahandler.xml");
        MainLogger.addHandler(file_handler);
        MainLogger.info("Info logger");
    }
}
```

```xml
<?xml version="1.0" encoding="windows-1250" standalone="no"?>
<!DOCTYPE log SYSTEM "logger.dtd">
<log>
<record>
  <date>2018-12-07T23:37:52.496243800Z</date>
  <millis>1544225872496</millis>
  <nanos>243800</nanos>
  <sequence>0</sequence>
  <logger>MainLogger</logger>
  <level>INFO</level>
  <class>Main</class>
  <method>main</method>
  <thread>1</thread>
  <message>Info logger</message>
</record>
</log>
```

```
gru 08, 2018 12:37:52 AM Main main
INFO: Info logger
```

Java – dzienniki

- Poniżej przedstawiono parametry konfiguracyjne obiektu typu FileHandler:

Własność konfiguracyjna	Opis	Wartość domyślna
`java.util.logging.FileHandler.level`	Poziom rejestracyjny obiektu `Handler`	`Level.ALL`
`java.util.logging.FileHandler.append`	Określa, czy handler powinien dopisywać dane do istniejącego pliku, czy dla każdego uruchomionego program otwierać nowy plik	`false`
`java.util.logging.FileHandler.limit`	Przybliżona maksymalna liczba bajtów, którą można zapisać w pliku, zanim zostanie otwarty kolejny (`0` oznacza brak limitu)	`0` (brak limitu) w klasie `FileHandler`, `50000` w konfiguracji domyślnego menedżera dzienników
`java.util.logging.FileHandler.pattern`	Wzorzec nazwy pliku dziennika. Zmienne wzorców zostały opisane w tabeli 11.2	`%h/java%u.log`
`java.util.logging.FileHandler.count`	Liczba dzienników w cyklu rotacyjnym	`1` (brak rotacji)
`java.util.logging.FileHandler.filter`	Klasa filtru, która ma zostać zastosowana	Brak filtru
`java.util.logging.FileHandler.encoding`	Kodowanie znaków	Kodowanie platformy
`java.util.logging.FileHandler.formatter`	Formater rekordów	`java.util.logging.XMLFormatter`

Java – dzienniki

- Domyślnie rekordy są filtrowane zgodnie z ich priorytetami.
- Każdy obiekt loggera oraz obiektu typu Handler może posiadać dodatkowy filtr rekordów.
- Definicja filtru polega na zaimplementowaniu interfejsu Filter i zdefiniowanu metody: boolean isLoggable(LogRecord record).
- Po przeanalizowaniu rekordu dziennika, stosując dowolne kryteria, zwracana jest wartość true, dla tych rekordów, które powinny trafić do dziennika.
- Przykładowo chcemy aby logowane były tylko rekordy INFO. W tym celu należy wywołać record.getMessage() i sprawdzić czy posiada słowo INFO.

Java – dzienniki

- Instalacja filtra w rejestratorze lub obiekcie typu Handler polega na wywołaniu metody setFilter.
- Należy pamiętać, że można używać tylko jednego filtra w danej chwili.

```
public class Main {
    private static final Logger MainLogger = Logger.getLogger("MainLogger");
    public static void main(String[] args) throws SecurityException, IOException {
        MainLogger.setLevel(Level.INFO);
        MainLogger.setFilter(new MainFilter());
        MainLogger.info("Wiadomość bez poziomu");
        MainLogger.info("Wiadomość poziomu INFO");
    }
}
```

```
gru 08, 2018 12:50:24 AM Main main
INFO: Wiadomość poziomu INFO
```

```
import java.util.logging.Filter;
import java.util.logging.LogRecord;

public class MainFilter implements Filter {
    public boolean isLoggable (LogRecord message)
    {
        String msg = message.getMessage();
        if(msg.contains("INFO"))
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}
```

Java – dzienniki

- Klasy ConsoleHandler oraz FileHandler tworzą zapisy dzienników w formie tekstu lub XML.
- Istnieje jednak możliwość zdefiniowana własnego formatu logów.
- W tym celu należy rozszerzyć klasę Formatter oraz przedefiniować metodę String format(LogRecord record).
- Informacje zawarte w rekordzie należy sformatować w dowolny sposób i zwrócić powstały łańcuch znaków.

Java – dzienniki

- Zalecenia do stosowania dzienników:
  1. W prostej aplikacji najlepiej wykorzystać jeden rejestrator. Dobrym pomysłem jest nadanie mu takiej nazwy jak nazywa się główny pakiet w aplikacji.
  2. Przy domyślnych ustawieniach wszystkie komunikaty od poziomu INFO są zapisywane w konsoli. W celu zmiany poziomu należy go zmienić dla rejestratora oraz obiektu typu Handler.
  3. Komunikaty poziomów INFO, WARNING i SEVERE są wypisywane w konsoli. Należy je zatem stosować dla komunikatów, które mają znaczenie dla użytkownika aplikacji.

Java – dzienniki

- Zalecenia do stosowania dzienników:
  3. Poziom FINE nadaje się doskonale dla komunikatów wykorzystywanych dla programistów. Wszędzie, gdzie programista chce wykorzystać System.out w celu wypisania komunikatu lepiej jest użyć logger.fine(„komunikat”).
  4. Dobrą praktyką jest logowanie wyjątków za pomocą loggerów.

Java – dzienniki

Constructor Summary

Constructors

Modifier	Constructor and Description
protected	`Logger(String name, String resourceBundleName)` Protected method to construct a logger for a named subsystem.

Method Summary

Methods

Modifier and Type	Method and Description
void	`addHandler(Handler handler)` Add a log Handler to receive logging messages.
void	`config(String msg)` Log a CONFIG message.
void	`entering(String sourceClass, String sourceMethod)` Log a method entry.
void	`entering(String sourceClass, String sourceMethod, Object param1)` Log a method entry, with one parameter.
void	`entering(String sourceClass, String sourceMethod, Object[] params)` Log a method entry, with an array of parameters.
void	`exiting(String sourceClass, String sourceMethod)` Log a method return.
void	`exiting(String sourceClass, String sourceMethod, Object result)` Log a method return, with result object.
void	`fine(String msg)` Log a FINE message.
void	`finer(String msg)` Log a FINER message.

Java – dzienniki

	Log a FINE message.
void	**finer**(String msg) Log a FINER message.
void	**finest**(String msg) Log a FINEST message.
static Logger	**getAnonymousLogger**() Create an anonymous Logger.
static Logger	**getAnonymousLogger**(String resourceBundleName) Create an anonymous Logger.
Filter	**getFilter**() Get the current filter for this Logger.
static Logger	**getGlobal**() Return global logger object with the name Logger.GLOBAL_LOGGER_NAME.
Handler[]	**getHandlers**() Get the Handlers associated with this logger.
Level	**getLevel**() Get the log Level that has been specified for this Logger.
static Logger	**getLogger**(String name) Find or create a logger for a named subsystem.
static Logger	**getLogger**(String name, String resourceBundleName) Find or create a logger for a named subsystem.
String	**getName**() Get the name for this logger.
Logger	**getParent**() Return the parent for this Logger.
ResourceBundle	**getResourceBundle**() Retrieve the localization resource bundle for this logger for the current default locale.
String	**getResourceBundleName**() Retrieve the localization resource bundle name for this logger.
boolean	**getUseParentHandlers**() Discover whether or not this logger is sending its output to its parent logger.

Java – dzienniki

void	**info**(String msg) Log an INFO message.
boolean	**isLoggable**(Level level) Check if a message of the given level would actually be logged by this logger.
void	**log**(Level level, String msg) Log a message, with no arguments.
void	**log**(Level level, String msg, Object param1) Log a message, with one object parameter.
void	**log**(Level level, String msg, Object[] params) Log a message, with an array of object arguments.
void	**log**(Level level, String msg, Throwable thrown) Log a message, with associated Throwable information.
void	**log**(LogRecord record) Log a LogRecord.
void	**logp**(Level level, String sourceClass, String sourceMethod, String msg) Log a message, specifying source class and method, with no arguments.
void	**logp**(Level level, String sourceClass, String sourceMethod, String msg, Object param1) Log a message, specifying source class and method, with a single object parameter to the log message.
void	**logp**(Level level, String sourceClass, String sourceMethod, String msg, Object[] params) Log a message, specifying source class and method, with an array of object arguments.
void	**logp**(Level level, String sourceClass, String sourceMethod, String msg, Throwable thrown) Log a message, specifying source class and method, with associated Throwable information.
void	**logrb**(Level level, String sourceClass, String sourceMethod, String bundleName, String msg) Log a message, specifying source class, method, and resource bundle name with no arguments.
void	**logrb**(Level level, String sourceClass, String sourceMethod, String bundleName, String msg, Object param1) Log a message, specifying source class, method, and resource bundle name, with a single object parameter to the log message.
void	**logrb**(Level level, String sourceClass, String sourceMethod, String bundleName, String msg, Object[] params) Log a message, specifying source class, method, and resource bundle name, with an array of object arguments.
void	**logrb**(Level level, String sourceClass, String sourceMethod, String bundleName, String msg, Throwable thrown) Log a message, specifying source class, method, and resource bundle name, with associated Throwable information.
void	**removeHandler**(Handler handler) Remove a log Handler.

Java – dzienniki

void	**setFilter**(**Filter** newFilter) Set a filter to control output on this Logger.
void	**setLevel**(**Level** newLevel) Set the log level specifying which message levels will be logged by this logger.
void	**setParent**(**Logger** parent) Set the parent for this Logger.
void	**setUseParentHandlers**(boolean useParentHandlers) Specify whether or not this logger should send its output to its parent Logger.
void	**severe**(**String** msg) Log a SEVERE message.
void	**throwing**(**String** sourceClass, **String** sourceMethod, **Throwable** thrown) Log throwing an exception.
void	**warning**(**String** msg) Log a WARNING message.

Kolejny wykład:

Deklaracje i nadpisywanie metod w języku Java.

Komentarze dokumentacyjne w języku Java.

DZIĘKUJĘ ZA UWAGĘ