**Wyższa Szkoła Informatyki Stosowanej i Zarządzania**

pod auspicjami Polskiej Akademii Nauk

**WYDZIAŁ INFORMATYKI**

**Kierunek INFORMATYKA**

Studia I stopnia (dyplom inżyniera)

# Język Java – wykład 2

dr inż. Łukasz Sosnowski
lukasz.sosnowski@wit.edu.pl
sosnowsl@ibspan.waw.pl
l.sosnowski@dituel.pl

www.lsosnowski.pl

# Cześć 1 – Obiekty, operatory, konwersje

# Przykład – przypomnienie

```
public class Car {
     protected int bodyType;
     protected int engineType;
     protected int engineCapacity;
     public Car(int bodyType,int engineType,int engineCapacity){
          this.bodyType = bodyType; this.engineType=engineType; this.engineCapacity=engineCapacity;
     }
     public double calculateFuelConsumtion(){
          double result=0.0d;
          //ciało metody
          return result;
     }
     public int getEngineCapacity() {
          return engineCapacity;
     }
     public void setEngineCapacity(int engineCapacity) {
          this.engineCapacity = engineCapacity;
     }
}
```

# Referencje do obiektów

- Zmienne typów obiektowych przekazywane są przez referencje

```
Car porsche1=new Car(1,1,3000);
Car porsche2 = porsche1;
porsche2.setEngineCapacity(2000);
```

- W przykładzie powyżej mamy tylko 1 obiekt!!!! Zmienna porsche2 otrzymała wartość referencji obiektu utworzonego linijkę wyżej. W tej sytuacji mamy 2 zmienne wskazujące na 1 ten sam obiekt. Stan obiektu po wykonaniu tego kodu będzie następujący:
  `bodyType=1, engineType=1, engineCapacity=2000`

- Parametry obiektowe metod i funkcji również są przekazywane przez referencję a zatem następuje wtedy modyfikacja stanu obiektu widoczna nie tylko lokalnie!

# Operator *new*

- Operator którego zadaniem jest utworzenie obiektu zadeklarowanej klasy
- Obiekt nie zawsze musi być utworzony ze względu na brak pamięci lub inną sytuację awaryjną, która zdarzyła się w konstruktorze.
- Operator ***new*** wywołuje konstruktor klasy, której obiekt jest tworzony, na podstawie doboru sygnatury (liczba i typy argumentów)
- Operator ***new*** zwraca referencję do utworzonego obiektu, która zostaje przypisana do zmiennej

```
Zmienna = new nazwa-klasy(lista-argumentów)
```

- W przypadku rzucenia wyjątkiem zmienna będzie mieć wartość taką na jaką została zainicjowana.

# Odzyskiwanie pamięci

- Proces zwalniania pamięci dokonywany jest w języku JAVA automatycznie
- W celu dokonania tej operacji muszą spełnione być 2 przesłanki: istnieją w pamięci nieużywane obiekty oraz istnieje konieczność odzyskania zajmowanej pamięci
- Operacja ta jest wiąże się z dodatkowym nakładem czasowym
- GarbageCollector to komponent odpowiedzialny za wykonywanie tego procesu zarządzany przez JVM. Istnieje jedynie możliwość zasugerowania uruchomienia czyszczenia pamięci, lecz w praktyce nie ma pewności czy czyszczenie nastąpi.
- Z założenia zarządzanie pamięcią dzieje się automatycznie, natomiast programista musi świadomie używać dobrych praktyk aby utylizacja pamięci była racjonalna

# Słowo kluczowe „this”

- Referencja do bieżącego obiektu dla którego np. wywoływana jest metoda
- Wewnątrz klasy daje możliwość odniesienia się do dowolnej składowej lub metody.
- Wewnątrz konstruktora daje możliwość wywołania innego konstruktora, np.. z inną liczbą parametrów
- Może być również wykorzystywany do przekazania referecji obiektu przez metodę, „return this;” co zwróci referencję do bieżącego obiektu zamiast np. „return new MyClass();”

```
public void setEngineCapacity(int engineCapacity) {
    this.engineCapacity = engineCapacity;
}
public Car(int bodyType){
    this(bodyType,1,2000);
}
```

# Operatory

- Symbol, który informuje kompilator, że należy wykonać określoną operację matematyczną lub logiczną
- Wyróżniamy kilka kategorii operatorów: arytmetyczne, bitowe, relacyjne, logiczne, specjalne.
- Najczęściej używany to operator przypisania:
  *zmienna = wyrażenie*
  gdzie typ zmiennej musi być zgodny z typem wyrażenia
- Przykłady operacji przypisania:
  x=y=z=10; → wszystkie 3 zmienne otrzymują wartość 10
  x+=10; → tożsame z zapisem x = x+10;
  x-=10; → tożsame z zapisem x =x-10;
  analogicznie x*=10, x|=10, itd..

# Operatory arytmetyczne

- Można stosować je dla każdego wbudowanego typu numerycznego a także typu *char*

| Operator | Opis |
|---|---|
| + | Dodawanie oraz operator unarny plus |
| - | Odejmowanie oraz operator unarny minus |
| * | mnożenie |
| / | dzielenie |
| % | Operacja modulo (reszta z dzielenia) |
| ++ | inkrementacja |
| -- | dekrementacja |

# Operatory relacyjne

- Operatory określające relację pomiędzy wartościami

| Operator | Opis |
|---|---|
| == | równe |
| != | różne |
| > | większe |
| < | mniejsze |
| >= | większe lub równe |
| <= | mniejsze lub równe |

# Operatory logiczne

- Operatory określające wartość logiczną wyrażenia (prawda/fałsz)

| Operator | Opis |
|---|---|
| & | koniunkcja (AND) |
| \| | alternatywa (OR) |
| ^ | różnica symetryczna (XOR) |
| && | warunkowa koniunkcja |
| \|\| | warunkowa alternatywa |
| ! | negacja |

# Tablica priorytetu operatorów

| Priorytet | Operatory | Opis | Przykład |
|---|---|---|---|
| 1 | ++, --, ~, !, rzutowanie | inkrementacja/dekrementacja przyrostkowa/ przedrostkowa, negacja bitowa, negacja logiczna | i++, ++i, i--,--i,~i,!i, y=(typ)x; |
| 2 | *, /, % | mnożenie, dzielenie, modulo | i*j,i/j, i % j |
| 3 | +, - | dodawanie, odejmowanie | i+j,i-j |
| 4 | <<, >>, >>> | przesunięcie bitowe w lewo, w prawo, w prawo bez znaku | I<<3,i<<4,i>>>4 |
| 5 | <, >, <=, >=, instanceof | mniejsze, większe, mniejsze i równe, większe i równe, instancja klasy | i<j,i>j,i<=j,i>=j, obj instanceof ExClass |
| 6 | ==, != | równość, różność | i==j, i!=j |
| 7 | & | bitowy iloczyn | i&j |
| 8 | ^ | bitowa różnica symetryczna (XOR) | i^j |

# Tablica priorytetu operatorów c.d.

| Priorytet | Operatory | Opis | Przykład |
|---|---|---|---|
| 9 | \| | bitowa suma | i\|j |
| 10 | && | koniunkcja warunkowa | (i==1)&&(j==9) |
| 11 | \|\| | alternatywa warunkowa | (i==0)\|\|(j==2) |
| 12 | ?: | operator warunkowy (if else) | i==j?1:0; |
| 13 | = | przypisanie | I=9; |

# Przykład

```
public class Rectangle {
     //Długość 1 boku
     private double a;
     //Długość 2 boku
     private double b;

     public Rectangle(double a,double b){
          this.a=a;
          this.b=b;
     }
     public double calculateArea(){
          return (a*b);
     }
     public void add(double add){
          this.a = a+add;
          this.b = b+add;
     }
     public static void main(String args[]){
          Rectangle p1 = new Rectangle(2.0,3.0);
          System.out.println(p1.toString());
          System.out.println("pole="+p1.calculateArea());
          p1.add(1.0);
          System.out.println(p1.toString());
          System.out.println("pole="+p1.calculateArea());
          p1.add(1.0);
          System.out.println(p1.toString());
          System.out.println("pole="+p1.calculateArea());
     }
}
```

```
public String toString(){
String item="";
item=item.concat("a=").concat(String.valueOf(a)).con
cat(", b=").concat(String.valueOf(b));
return item;
}
```

```
WYNIK:
a=2.0, b=3.0
pole=6.0
a=3.0, b=4.0
pole=12.0
a=4.0, b=5.0
pole=20.0
```

# Automatyczna konwersja typów

- Dotyczy dwóch zmiennych o różnych ale zgodnych typach takich, że typ docelowy jest większy
- Polega na zamianie wartości po prawej stronie na wartość typu po lewej stronie, np.

```
int i=10;
float j;
j=i;
```

- JAVA zakłada automatyczną konwersję dla typów numerycznych całkowitych z krótszych na dłuższe, np.. int → long
- JAVA wykonuje automatyczną konwersję również pomiędzy typami całkowitoliczbowymi a zmiennoprzecinkowymi **lecz nie odwrotnie**

# Rzutowanie typów niezgodnych

- Rzutowanie jest jawnym żądaniem konwersji wskazanych typów wyrażane jest poprzez następującą konstrukcję:
  `(typ-docelowy) (wyrażenie)`
- „Typ-docelowy” określa typ, do którego należy wykonać konwersję
- Jeśli reprezentacja typu docelowego nie jest wystarczająco pojemna może dojść do utraty informacji!!!
- Rzutowanie pomiędzy typami zmiennoprzecinkowymi a całkowitoliczbowymi jest możliwe lecz należy to robić świadomie. Dodatkowo należy pamiętać o utracie informacji po przecinku
- Przykłady:

```
double x,y; byte b; int i; char ch;
x=10.0; y=4.0; i=(int)(x/y); i = 100; b = (byte)i; b=88;  ch=(char)b;
```


# Cześć 2 – Instrukcje sterujące

# Instrukcja warunkowa „*if*”

- Składnia dla pojedynczej instrukcji:

```
if(warunek) instrukcja
else instrukcja;
```

gdzie klauzula „*else*” jest opcjonalna

- Instrukcja może również dotyczyć bloku kodu zarówno w części „*if*” jak i „*else*”. Składnia:

```
if(warunek){
  sekwencja instrukcji
}else{
  sekwencja instrukcji
}
```

# Instrukcja warunkowa „*if*” c.d.

- Instrukcja może również być zbudowana z wielu alternatywnych warunków, które sprawdzane są tylko jeśli poprzedni jest fałszywy. Składnia:

```
if(warunek1){
    sekwencja instrukcji
  }else if(warunek2){
    sekwencja instrukcji
  }else if(warunek3){
    sekwencja instrukcji
  }else if(warunek4){
    sekwencja instrukcji
  }else{
    sekwencja instrukcji
  }
```

- Klauzula „else” jest opcjonalna. Jeśli żaden warunek nie będzie spełniony, nie wykona się żadna instrukcja

# Przykład instrukcji warunkowej „*if*”

```
int i=100;

if(i==0)
    System.out.println("i="+i);
else if(i==1)
    System.out.println("i="+i);
else if(i>1) {
    System.out.println("i>1");
}
else if(i<0) {
    System.out.println("i<0");
}
else {
    System.out.println("w p.p.");
}
```

# Instrukcja „switch”

- Instrukcja umożliwiająca tworzenie rozgałęzień w kodzie.

```
switch(wyrażenie){
    case stała1:
      sekwencja instrukcji
      break;
    case stała2:
      sekwencja instrukcji
      break;
    default:
      sekwencja instrukcji
  }
```

- „Wyrażenie” w wersjach starszych niż JDK7 może być typu `byte, short, int, char` natomiast od JDK7 może być również typu `String`
- Klauzula `default` jest opcjonalna. Odpowiada za przejęcie sterowania w przypadku braku spełnienia warunków w poszczególnych klauzulach `case`

# Instrukcja „switch” c.d.

- Każda wartość w instrukcji `case` musi być stała i unikalna
- Instrukcja `break` powoduje przeniesienie sterowania poza instrukcję `case` oznaczoną nawiasem końcowym
- Jeśli instrukcja break nie wystąpi, wtedy kolejne instrukcje z kolejnych „case” będą wykonywane po kolei
- Możliwe jest stosowanie pustych instrukcji case jak również zagnieżdżonych

```
int i=1;
switch(i) {
case 1:
case 2:
     System.out.println("1 i 2");
case 3:                                    →    ”1 i 2”
case 4:                                         ”3 i 4”
     System.out.println("3 i 4");
     break;
case 5:
     System.out.println("5");
     break;
}
```

# Pętla „for”

- Instrukcja pozwalająca na iteracje po kolejnych wartościach zmiennej lub kolekcji
- Podstawowa składnia dla pojedynczej instrukcji:

```
for(inicjalizacja;warunek;iteracja) instrukcja
```

i dla bloku instrukcji:

```
for(inicjalizacja;warunek;iteracja){
  instrukcje
}
```

- Przykład podstawowy:

```
for(int i=0;i<10;i++) {
     System.out.println("i="+i);
}
```

- Pętla rozszerzona daje możliwość iterowania po kolejnych obiektach kolekcji. Składnia:

```
for(typ zmienna:kolekcja){
  instrukcje
}
```

# Pętla „for” c.d.

- Przykład dla pętli rozszerzonej:

```
double[] dbArr = new double[] {1.1,2.2,3.3,4.4,5.5}; → tablica 1 wymiarowa typu double (będzie na dalszych wykładach)
for(double d:dbArr) {
      System.out.println("d="+d);
}
```

- Wynik na konsoli:

```
d=1.1
d=2.2
d=3.3
d=4.4
d=5.5
```

- Specyficzne warianty pętli „for”
  - Pętla nieskończona: `for(;;) instrukcja;`
  - Pętla bez ciała:`int sum=0;  for(int i=1;i<=20;sum+=i++);`
  - Pętla bez wyrażenia iteracyjnego: `for(int i=0;i<5;){ instrukcja; i++;}`
  - Pętla z dwoma zmiennymi sterującymi: `for(int i=0,j=10;i<j;i++,j--) instrukcja;`

# Pętla „while”

- Ogólna postać instrukcji:

```
while(warunek) instrukcja;
```

- Pętla wykonuje kolejne przebiegi dopóki warunek jest prawdziwy
- Pętla `while(true) instrukcja;` jest pętlą nieskończoną
- Pętla przed wykonanie pierwszego przebiegu sprawdza wartość warunku
- Przykład:

```
int i=0;
while(i<5) {
     System.out.println("i="+i);
     i++;
}
```

# Pętla „do-while”

- Ogólna postać instrukcji:

```
do{
  instrukcje;
}while(warunek);
```

- Pętla wykonuje sprawdzanie warunku na końcu przebiegu
- Ten typ pętli wykonuje zawsze minimum 1 przebieg
- Pętla wykonywana do momentu uzyskania wartości fałsz przez warunek
- Przykład:

```
int i=0;
do {
     System.out.print("i="+i);    →    i=0
     i++;
}while(i<1);
```


# Cześć 3 – Łańcuchy znaków

# Łańcuchy znaków w JAVA

- Języku JAVA typem używanym do reprezentacji łańcuchów znaków jest *String*
- *String* jest typem obiektowym
- Do utworzenia obiektu klasy *String* możemy użyć operatora new lub bezpośrednio przypisać wartość łańcucha znaków;
- Zmienna deklarowana w programie przechowuje referencję do obiektu zawierającego zadany łańcuch znaków
- Przykłady

```
String tmp = new String("Testowy napis");
String tmp2 = new String(tmp);
String tmp3 = "Testowy napis";
String tmp4 = "Testowy napis";
```

# Wybrane metody klasy String

- *boolean equals(str)* – zwraca wartość logiczną porównania łańcuchów zapisanych w obiekcie na którym wywołana jest metoda oraz przekazanym z argumencie
- *int lenght()* - metoda zwracająca liczbę znaków łańcucha
- *char charAt(index)* – zwraca pojedynczy znak z pozycji index, która rozpoczyna się od 0
- *int compareTo(str)* – zwraca wartość ujemną jeśli łańcuch pierwszy jest mniejszy niż przekazany w argumencie, zero w przypadku równości łańcuchów i wartość większą od zera w p.p.
- *int indexOf(str)* – wyszukuje podłańcuch przekazany w argumencie. Zwraca index pierwszego wystąpienia podłańcucha lub -1 w przypadku braku występowania

# Wybrane metody klasy String c.d.

- *int lastIndexOf(str)* – wyszukuje podłańcuch przekazany w argumencie. Zwraca *index* ostatniego wystąpienia podłańcucha lub -1 w przypadku braku występowania
- UWAGA: Łańcuchy porównujemy poprzez metody *equals* lub *compareTo* a nie operatory!!! Operator porównuje wartość referencji do obiektu a nie obiekty!!!
- Przykład

```
String tmp = new String("Testowy napis");
String tmp2 = new String(tmp);
String tmp3 = "Testowy napis";
System.out.println("tmp="+tmp);
System.out.println("tmp2="+tmp2);
System.out.println("tmp3="+tmp3);


System.out.println(tmp==tmp2?"Takie same":"Różne");          →    tu porównujemy referencje
System.out.println(tmp.equals(tmp2)?"Takie same":"Różne");   →    tu porównujemy łańcuchy które są obiektami

tmp=Testowy napis
tmp2=Testowy napis
tmp3=Testowy napis
Różne
Takie same
```

# Modyfikowanie łańcuchów znaków

- Zawartość obiektu klasy String jest niezmienna!!!
- W celu modyfikacji wartości łańcucha musimy utworzyć nowy obiekt
- Operator „+” dla obiektów klasy String istnieje, lecz wykonuje on w tle operacje typu: `new StringBuilder(s).append("a").toString();` Operacja ta jest bardzo nieefektywna.
- Do pracy na zmiennych łańcuchach dostępna jest klasa StringBuilder
- Zamiast: `String test=”Ala”; test = test+”ma kota”;`
- używamy `StringBuilder sB = new StringBuilder(„Ala); sB.append(”ma kota”); String test = sB.toString();`
- W przypadku wielu milionów operacji w pętli różnica złożoności czasowej może być o rząd wielkości lub więcej


# Cześć 4 – Tablice

# Tablice ogólnie

- Kolekcja zmiennych tego samego typu, do których odwołujemy się za pomocą wspólnej nazwy oraz indeksu wskazującego na pozycję w tablicy
- Tablice są obiektami
- Tablice dają możliwość łatwego sortowania przechowywanych danych
- Tablice ułatwiają przetwarzanie przechowywanych danych, wyliczanie wartości pochodnych względem przechowywanych danych, np. wartość średnia wartości
- Tablice mogą być jedno lub wielowymiarowe adresowane taką liczbą indeksów jaka jest wymiarowość
- Ogólna składnia: `typ nazwa[][]..[] = new typ[r1][r2]..[rN];`

# Tablice jednowymiarowe

- Deklaracja tablicy: typ nazwa-tablicy[] = new typ[rozmiar];
- Elementy tablicy mają zadeklarowany wspólny typ
- Liczba elementów tablicy definiuje jej rozmiar
- Przykład:
  - `int arr[] = new int[5];`
  - `int arr[]; arr = new int[6];`
  - `int arr[] = new int[]{3,4,21,2,1};`
- Do elementów tablicy odwołujemy się poprzez nazwę oraz indeks, np. `arr[0]` to odwołanie do elementu o indeksie 0;
- Liczba elementów wymiaru tablicy zwracana jest poprzez składową `length`
- Przeglądanie elementów tablicy najlepiej zaimplementować przy użyciu pętli

# Tablice dwuwymiarowe

- Dwuwymiarowa tablica jest tablicą tablic. Posiada dwa niezależne indeksy. Pierwszy dotyczy numeru wiersza, drugi numeru kolumny
- Deklaracja: `typ nazwa[][];`
- Powołanie obiektu: `nazwa = typ[r1][r2];`
- Powołanie obiektu i inicjalizacja:
  `nazwa = typ[r1][r2] {new typ{…},new typ{…}};`
- Odwołanie do elementu poprzez `nawa[0][0];` → zwraca wartość int
- Odwołanie do tablicy jednowymiarowej `nazwa[1]` → zwraca tablicę 1 wymiarową
- Składowa *length* może być wywołana na dowolnym wymiarze

# Tablice - możliwości

- Deklarowanie tablic ma swoją alternatywną wersję, np..
  typ[] nazwa-zmiennej;
  typ[] nazwa1, nazwa2, nazwa3;
  typ nazwa1[], nazwa2[], nazwa3[];
  typ[][] nazwa -zmiennej;
- java.util.Arrays – klasa dająca szereg użytecznych narzędzi do pracy z tablicami, np.. sort(arr), toString(arr), compare, equals
- Przykład sortowana: `int arrInt[] = new int[] {10,5,3,13,2,19,22,21,22,1}; Arrays.sort(arrInt);`
- Ustawianie wartości konkretnego indeksu: mp. nazwa[0][2]=1;

# Tablice - przykład

```
int arrInt[] = new int[] {10,5,3,13,2,19,22,21,22,1};
System.out.println(""+Arrays.toString(arrInt));
Arrays.sort(arrInt);
System.out.println(""+Arrays.toString(arrInt));
```

```
Wynik:  [10, 5, 3, 13, 2, 19, 22, 21, 22, 1]
        [1, 2, 3, 5, 10, 13, 19, 21, 22, 22]
```

```
int arrInt[][] = new int[][] {new int[] {10,5,3,13,2,19,22,21,22,1},new int[] {0,0,0,0,0,1,1,1,1,1}};

System.out.println(""+Arrays.toString(arrInt[0]));
System.out.println(""+Arrays.toString(arrInt[1]));
Arrays.sort(arrInt[0]);
Arrays.sort(arrInt[1]);

System.out.println(""+Arrays.toString(arrInt[0]));
System.out.println(""+Arrays.toString(arrInt[1]));
```

```
Wynik:  [10, 5, 3, 13, 2, 19, 22, 21, 22, 1]
        [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]
        [1, 2, 3, 5, 10, 13, 19, 21, 22, 22]
        [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]
```

# Cześć 5 – komentarze w JAVA

# Komentarze w kodzie JAVA

- Komentarz jednoliniowy pozwalający na wprowadzenie dowolnego tekstu  jedno liniowego po sekwencji znaków „//”

```
//Długość 1 boku
private double a;
```

- Komentarz wieloliniowy pozwalający na wprowadzenie dowolnego tekstu zawartego pomiędzy znacznikami „/*” oraz „*/

```
/**
 * Metoda licząca pole powierzchni prostokąta
 * @return
 */
public double calculateArea(){
    return (a*b);
}
```

- Komentarz dokumentacyjny, służący do automatycznego generowania dokumentacji kodu w postaci *javadoc*

# Komentarze dokumentacyjne – wybrane elementy

- @author – identyfikuje autora kodu (np. klasy)
- @param – dokumentuje parametr
- @return – dokumentuje wartość zwracaną przez metodę
- @see – tworzy łącze do innego tematu dokumentacji
- @deprecated – informuje że element programu jest przestarzały
- @exception – definiuje wyjątek rzucany przez metodę, konstruktor
- @since – określa wersję w której wprowadzono daną zmianę
- @version – określa wersję elementu programu

```
/**
 *
 * @author Łukasz Sosnowski
 *
 */
public class Rectangle {...
```

# Przykład

```
/**
 * @author Łukasz Sosnowski
 */
public class Rectangle {
     //Długość 1 boku
     private double a;
     //Długość 2 boku
     private double b;
     public Rectangle(double a,double b){
          this.a=a;
          this.b=b;
     }
     /**
      * Metoda licząca pole powierzchni prostokąta
      * @return
      */
     public double calculateArea(){
          return (a*b);
     }
     /**
      * Metoda dodająca zadaną wartość
      * do obu boków prostokąta
      * @param add
      */
     public void add(double add){
          this.a = a+add;
          this.b = b+add;
     }
}
```

# Podsumowanie

- Podstawowa wiedza o operatorach
- Priorytety operatorów w JAVA
- Pojęcie hermetyzacji w teorii i praktyce
- Konwersja typów jawna i automatyczna
- Instrukcje sterujące w języku JAVA
- Łańcuchy znaków w JAVA
- Tablice jedno i wielowymiarowe w języku JAVA
- Rodzaje komentarzy w kodzie

**Wyższa Szkoła Informatyki Stosowanej i Zarządzania**

pod auspicjami Polskiej Akademii Nauk

**WYDZIAŁ INFORMATYKI**

**Kierunek INFORMATYKA**

Studia I stopnia (dyplom inżyniera)

---

# Dziękuję za uwagę!