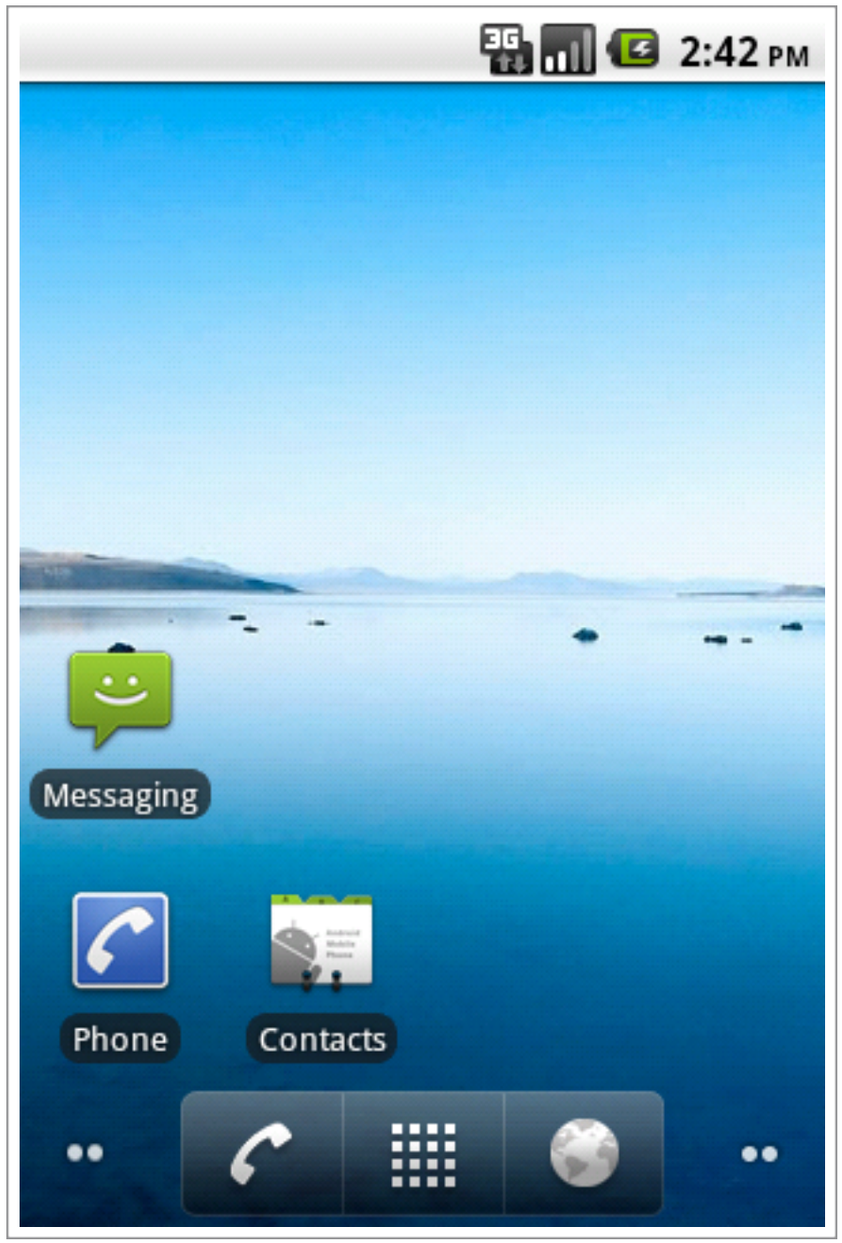
Programowanie urządzeń mobilnych

dr inż. Andrzej Grosser
na podstawie wykładu
dr inż. Juliusza Mikody

Najważniejsze platformy mobilne

- Android
- iOS
- Windows 10
- Fuchsia
- Sailfish OS
- Tizen
- Ubuntu Touch

Android

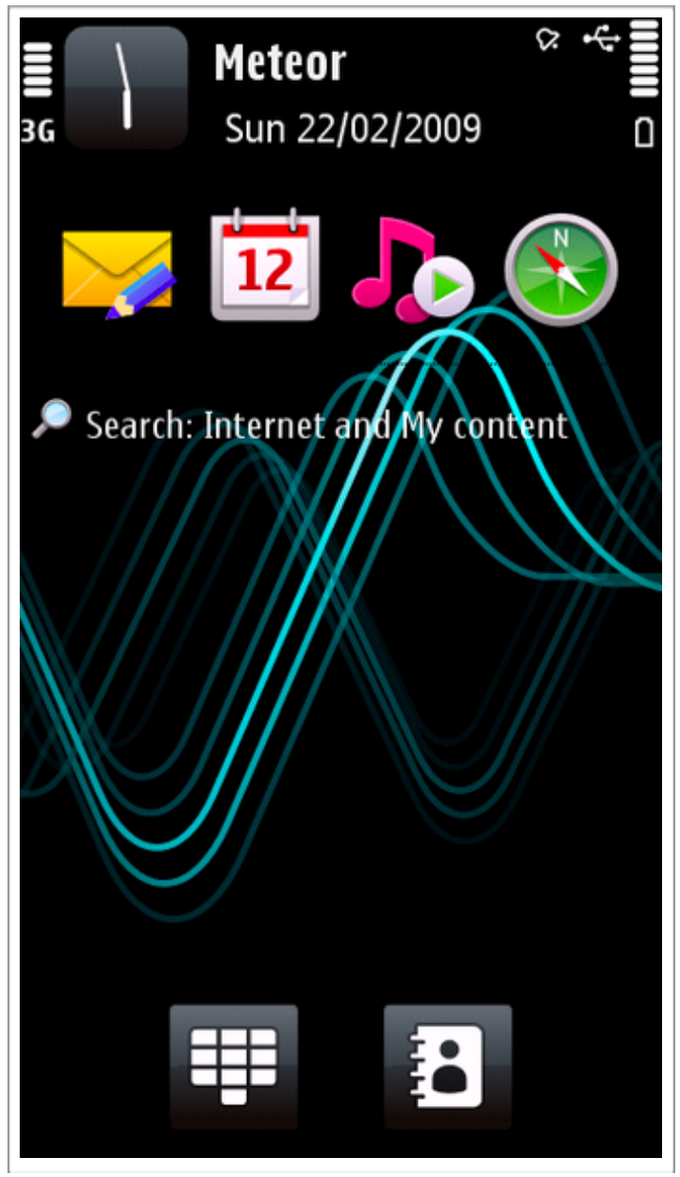


- Producent systemu: Open Handset Alliance
- Producenci urządzeń: Samsung, Motorola, HTC i inne
- Programowanie: Java, Kotlin, wirtualna maszyna Dalvik (ART), C++ (tylko fragmenty aplikacji wymagające dużej wydajności)

iOS

- Producent systemu: Apple
- Urządzenia: iPhone, iPod touch, iPad.
- Języki programowania: Objective-C, Swift, iOS SDK.
- Bazuje na systemie operacyjnym Mac OS X 10.5 i jądrze Darwin.

Sailfish OS



- Producent systemu: Sailfish Alliance
- Producenci urządzeń: Jolla
- Programowanie: C++, QML.

Tizen

- Producent systemu: Samsung, Intel
- Dostępny jedynie na urządzenia Samsunga – Galaxy Gear, Samsung Z (planowany)
- Programowanie: C++, Tizen SDK

Język Kotlin

- Projekt rozpoczął się w roku 2011 w JetBrains.
- W 2017 wraz z premierą Android Studio 3.0 jest wspierany na platformie Android.
- Jest językiem:
  - statycznie typowanym wysokiego poziomu,
  - zgodnym z językiem Java,
  - przenośnym między platformami.

Język Java

- Język Java powstał w roku 1995 w firmie SUN Microsystems
- Java jest językiem:
  - wysokiego poziomu,
  - w pełni obiektowym,
  - przenośnym między platformami.

Typy proste

- boolean (Boolean) false
- char - 2 bajty (Character) '\u0000'
- byte - 1 bajt (Byte) 0
- short - 2 bajty (Short) 0
- int - 4 bajty (Integer) 0
- long - 8 bajtów (Long) 0L
- float - 4 bajty (Float) 0.0f
- double - 8 bajtów (Double) 0.0d
- void (Void)

Typy proste

- Brak typów unsigned.
- Wielkość typów niezależna od platformy.
- Każdy typ prosty ma odpowiednik w postaci klasy „opakowującej” typ prosty.
- Java udostępnia dwa typy przeznaczone do obliczeń wysokiej precyzji: **BigInteger** oraz **BigDecimal**. Nie są to jednak typy proste, ale klasy.
- Typy proste użyte jako pola klasy są automatycznie inicjowane wartościami domyślnymi, natomiast zmienne lokalne nie są automatycznie inicjowane.

Klasy opakowujące

sposób na referencje do typów podstawowych

- Klasa opakowująca typ prosty pozwala tworzyć obiekty tego typu oraz referencje do tych obiektów.
- Klasy opakowujące są klasami finalnymi - wartości obiektów tych klas nie mogą być zmieniane. Każda modyfikacja wartości powoduje utworzenie nowego obiektu.

```
Integer a = 1;

Integer b = a;

Integer c = b; // a, b i c wskazują na ten sam obiekt

a = 5; // tworzony jest nowy obiekt inicjowany wartością 5 (a=5,b=c=1)

b++; // tworzony jest nowy obiekt inicjowany wartością 2 (a=5,b=2,c=1)

// a, b, c wskazują na inne obiekty!
```

Metody klasy String

co można zrobić ze Stringiem nie męcząc się zbyt wiele

Operator +

- Konkatenacja Stringów

```
String napis = "Ala ";
napis = napis + "ma kotka ";
napis += "i pieska";
// Ala ma kotka i pieska
```

- Doklejanie liczb to Stringów

```
String str2, str = "napis";
str2 = str + 10; //napis10
str2 = str + (10+10); //napis20
```

Metody klasy String

```
String s="aBCDe";
s.length(); //5
s.indexOf('C'); //2
s.indexOf('Z'); //-1
s.toLowerCase(); //"abcde"
s.toUpperCase(); //"ABCDE"
s.substring(0,2); //"aB"
s.charAt(2); //'C'
```

Porównywanie stringów

```
String s1 = "abc" + 12; String s2 = "abc12";
```

Porównanie referencji:

```
System.out.println( s1 == s2 ); // false
```

Sprawdzenie czy stringi są równe:

```
System.out.println( s1.equals(s2) ); // true
System.out.println( s1.equals("ABC12") ); // false
```

Obliczenie różnicy między stringami:

```
System.out.println( s1.compareTo(s2) ); // 0
System.out.println( s1.compareTo("ABC12")); // 32
```

Tworzenie klas i obiektów

definiowanie nowych typów danych

- Utworzenie klasy:

```
class NazwaKlasy { /* metody i pola składowe */ }
```

- Utworzenie instancji klasy (obiektu):

```
NazwaKlasy a = new NazwaKlasy();
```

Słowo **new** powoduje utworzenie nowego obiektu typu `NazwaKlasy`
`a` to referencja do tego obiektu.

```
NazwaKlasy b = a; // a i b wskazują na ten sam obiekt!
```

Nie ma konieczności jawnego zwalniania pamięci przydzielonej przez new!

Atrybuty

pola składowe klasy

```
class A {
    int p1;
    // typ prosty zainicjowany wartością domyślną 0
    int p2 = 10; // typ prosty zainicjowany liczbą 10
    Integer p3;
    // nieprzydzielona pamięć dla obiektu - wartość null
    Integer p4 = new Integer(10);
    // referencja do obiektu zainicjowanego wartością 10
}
```

Dostęp do pól składowych odbywa się poprzez podanie nazwy obiektu oraz nazwy pola połączonych kropką.

```
A a = new A();
a.p1 = 20;
```

Metody

funkcje składowe klasy

```
class A{
    void f1(){ /* instrukcje */ }
    int f2(int x, Integer y, B b){ /* instrukcje */
    B f3(){ /* instrukcje */ }
}
```

- Aby wysłać wiadomość (żądanie wykonania metody) do obiektu należy użyć nazwy obiektu oraz nazwy metody połączonych kropką

```
A a = new A();
a.f1(); // wysłanie do obiektu a żądania wykonania metody f1
int i = a.f2(10,20,new B());
B b = a.f3();
```

W Javie funkcje mogą być tworzone tylko wewnątrz klasy!

Funkcje nie mogą mieć domyślnych wartości parametrów.

`this`

dostęp do atrybutów wewnątrz metod

- Wewnątrz metod klasy dostępna jest zmienna **`this`** będąca referencją do obiektu na rzecz którego wywoływana jest metoda.
- Dostęp do pól klasy możliwy jest za pomocą zmiennej **`this`** lub bezpośrednio poprzez nazwę pola klasy.

```
class A {
    int p;
    void f1(){
        p = 1;
        this.p = 2;
        System.out.println(p); // 2
        int p = 3;
        System.out.println(p); // 3
        System.out.println(this.p); // 2
    }
}
```

`System.out.print( )` - wypisanie na ekran wartości przekazanej jako parametr.
`System.out.println( )` - wypisanie i przejście do nowej linii.

C++ vs. Java

co wolno w C++, a w Javie nie

- przesłanianie zmiennych - C++ OK, Java błąd

```
{
    int x = 12;
    {
        int x = 96; // błąd
    }
}
```

- użycie niezainicjalizowanych zmiennych - C++ OK, Java błąd podczas kompilacji!

```
int x;
int y = x; // błąd
```

- zwalnianie pamięci - w Javie nie ma konieczności jawnego zwalniania przydzielonej pamięci. Zajmuje się tym odśmiecacz (ang. *garbage collector*), który zlicza referencje do obiektu i jeśli już żadna nie wskazuje na obiekt zwalnia pamięć. Brak wycieków pamięci!

Instrukcje sterujące

Instrukcje warunkowe:

```
if ( warunek ) instrukcje;
else instrukcje;
switch ( zmienna ) {
    case wartość: instrukcje; break;
    default: instrukcje;
}
```

Pętle:

```
do { instrukcje } while ( warunek )
while ( warunek ) { instrukcje }
for ( inicjalizacja; warunek; krok ) { instrukcje }
```

W języku Java warunek musi być wyrażeniem typu logicznego.
Nie jest wykonywana konwersja typów.

```
if ( i = 10 ) // błąd
int a = 0;
if ( a ) // błąd
```

Witaj obiektowy świecie!

pierwszy program w Javie

```
public class Witaj {
    public static void main(String[] args) {
        /* Tu program rozpoczyna swoje działanie*/
        System.out.print("Witaj obiektowy świecie!");
    }
}
```

- Nazwa pliku taka sama jak nazwa klasy!

Witaj.java

Kompilacja i uruchamianie

Pliki źródłowe w języku Java:

- nazwa pliku identyczna jak nazw klasy publicznej
- rozszerzenie .java

Kompilacja:

- javac Witaj.java

Pliki wykonywalne:

- rozszerzenie .class

Uruchamianie programu:

- java Witaj

Programy stworzone w Jawie podlegają kompilacji do bytecode'u i są uruchamiane na maszynie wirtualnej Javy

Zmienne tablicowe

- tablice jednowymiarowe

```
int tab[]; // deklaracja tablicy
int [] tab2; // równoważna deklaracja
tab = new int[5]; // przydział pamięci
tab[2] = 5; // odwołanie do elementu tablicy
int tab3[] = {1, 2, 3, 4, 5};
             // inny sposób przydziału pamięci
             // wraz w inicjacją tablicy
```

- tablice wielowymiarowe

```
int tablica2d[][];
tablica2d = new int [10] [10];
tablica2d[2][4] = 12;
```

Tablice c.d.

Java "pamięta" rozmiar tablicy. Można go odczytać w następujący sposób:

```
int n = tab.length;
```

rozmiary tablicy wielowymiarowej:

```
int n = tablica2d.length;

int m = tablica2d[0].length;
```

Przypisanie tablicy to przypisanie referencji:

```
int b [] = new int[10];

int a [] = b;
```

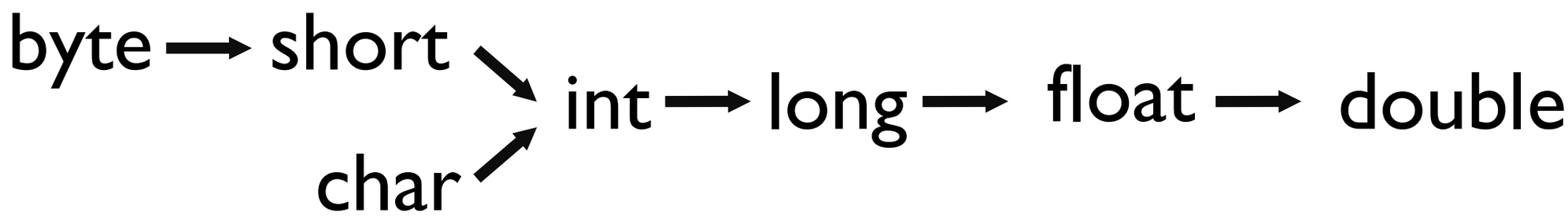
Konstruktor klasy

```
class Prostokat
{
    int a;
    int b;
    int pole;
    Prostokat( int aa , int bb )
    {
        a = aa;
        b = bb;
        pole = a*b;
    }
}
```

- Możliwe jest utworzenie kilku konstruktorów różniących się liczbą lub typem przyjmowanych parametrów.
- Konstruktor nie przyjmujący żadnych wartości to konstruktor domyślny.
- Jeśli nie zostanie zdefiniowany żaden konstruktor kompilator automatycznie utworzy konstruktor domyślny.

Awansowanie typów prostych

wywołanie metod z parametrami innych typów niż zdefiniowane



W przypadku typów prostych, jeśli wywołamy metodę (również konstruktor) z innym parametrem niż zdefiniowany kompilator spróbuje awansować typ zmiennej według powyższego schematu.

```
class A{
  void f(double x){ System.out.println("f(double) "+x); }
  void f(int x) { System.out.println("f(int) "+x); }
}
A a = new A();
a.f(10); // f(int) 10
float x1 = 10.0f;
a.f(x1); // f(double) 10.0
char x2 = 'a';
a.f(x2); // f(int) 97
```

Dziedziczenie w Javie

- Dziedziczenie w Javie realizowane jest za pomocą słowa **extends**

```
class KlasaPochodna extends KlasaBazowa { /*...*/ }
```

- Przykład dziedziczenia klasy Pracownik i Czlowiek, Kierownik i Pracownik

```
class Czlowiek {
    String imie;
    String nazwisko;
    boolean plec;
}
class Pracownik extends Czlowiek {
    String stanowisko;
    double pensja;
    String dzial;
}
class Kierownik extends Pracownik {
int liczbaPracownikow;}
```

super

jak skłonić Pracownika, żeby zachowywał się jak Człowiek?

Dostęp do przesłoniętych metod klasy bazowej wewnątrz metod klasy pochodnej realizowany jest za pomocą słowa **super**

```
class Czlowiek {
    String imie;
    String nazwisko;
    boolean plec;
    String przedstawSie(){
        return imię + " " + nazwisko + " " + (plec ? "kobieta" : "mezczyzna");
    }
}
class Pracownik extends Czlowiek {
    String stanowisko;
    double pensja;
    String dzial;
    String przedstawSie(){
        // wywołanie metody z klasy Człowiek i "doklejenie" atrybutów z klasy Pracownik
        return super.przedstawSie() +" "+ stanowisko +" "+ pensja +" "+ dzial;
    }
}
```

Jak skłonić Kierownika, żeby zachowywał się jak Człowiek?

```
super.super.przedstawSie(); // Niemożliwe!
```

Hermetyzacja

w Javie

W języku Java hermetyzacja jest realizowana za pomocą modyfikatorów dostępu.

Istnieją cztery poziomy dostępu

- publiczny `public`,
- chroniony `protected`,
- pakietowy - domyślny, brak jawnej specyfikacji dostępu,
- prywatny `private`.

Modyfikatory dostępu umieszcza się przed deklaracją atrybutu, metody lub klasy. Dla atrybutów i metod można stosować wszystkie modyfikatory dostępu, dla klas jedynie public (lub domyślnie - pakietowy).

public

dostępne dla wszystkich

- Dostęp do pól i metod publicznych mają wszyscy, w szczególności programiści korzystający z danej klasy.
- Zgodnie z zasadami hermetyzacji żadne pola składowe klasy nie powinny być publiczne.
- Klasa publiczne - tylko jedna w pliku!

```
class A{
    public int x;
}

A a = new A();
a.x = 10; // możliwy zapis atrybutu
int y = a.x; // możliwy odczyt atrybutu
```

protected

dostępne tylko wewnątrz pakietu oraz dla potomków

- Dostęp do pól i metod jest możliwy jedynie z metod klas należących do danej klasy, klas należących do tego samego pakietu lub klas dziedziczących po danej klasie.

```
class A{
    protected int x;
    void f1(){
        this.x = 10; // OK
        int i = this.x; // OK
    }
}

class B extends A{
    void f2(){
        this.x = 10; // OK
        int i = this.x; // OK
    }
}

A a = new A();
a.f1(); // OK tylko wewnątrz pakietu
```

pakietowy

dostępne tylko wewnątrz pakietu

- Dostęp do pól i metod jest możliwy jedynie z metod danej klasy oraz klas należących do tego samego pakietu.

```
class A {
    int x;
    void f1(){
        this.x = 10; // OK
        int i = this.x; // OK
    }
}
class B extends A {
    void f2(){
        this.x = 10; // OK tylko wewnątrz pakietu
        int i = this.x; // OK tylko wewnątrz pakietu
    }
}

A a = new A();
a.x = 10; // OK tylko wewnątrz pakietu
int y = a.x; // OK tylko wewnątrz pakietu
```

private

dostępne tylko dla danej klasy

- Dostęp do pól i metod jest możliwy jedynie z metod tej samej klasy.

```
class A {
    private int x;
    void f1(){
        this.x = 10; // OK
        int i = this.x; // OK
    }
}
class B extends A {
    void f2(){
        this.x = 10; // Błąd!
        int i = this.x; // Błąd!
    }
}

A a = new A();
a.x = 10; // Błąd
int y = a.x; // Błąd
```

Metody dostępowe

...jak modyfikować to co niedostępne

```
class A{
    private int x;

    public int getX() { return this.x; }
    public void setX(int i) { this.x = i; }
}

A a = new A();

//a.x = 10; // błąd!
a.setX(10);

//int i = a.x; // błąd!
int j = a.getX();
```

this

...kolejne zastosowania

- wywołanie konstruktora wewnątrz innego konstruktora:

```
class A{
    private double y;
    private int x;
    A(int i){ this.x = i; }
    A(int i, double d){
        this(i);
        this.y = d;
    }
}
```

- zwrócenie referencji do obiektu, na rzecz którego wywołana jest metoda:

```
class A{
    private int x;
    A(int i) { x=i; }
    int getX() { return x; }
    A inkrementuj() {
        x++;
        return this;
    };
}
A a = new A(10);
a.inkrementuj().inkrementuj().inkrementuj();
System.out.println( a.getX() ); // 13
```