# Rozwój oprogramowania w R: OOP

Mateusz Staniak

*Rozwój oprogramowania w R*

Instytut Matematyczny UWr, semestr letni 2020

Uniwersytet Wrocławski


# Programowanie obiektowe

# Podstawowe pojęcia (przypomnienie)

Uniwersytet Wrocławski

1. **Klasa** – "szablon" / "opis", mówiący, opisujący stan (atrybuty) i możliwe operacje (metody).
2. **Obiekt** – instancja klasy.
3. **Metoda** – funkcja (tradycyjnie związana z klasą).
4. **Dziedziczenie** – sposób rozszerzania klasy – klasa może wykorzystywać zmienne i metody klasy, po której dziedziczy (zmieniając ich zachowanie).
5. **Enkapsulacja** – ukrywanie danych / implementacji przed kodem, który używa obiektów danej klasy.
6. **Polimorfizm** – zróżnicowanie zachowania w zależności od typu obiektów, na których operuje.
7. **Agregacja**, kompozycja… - sposoby tworzenia bardziej złożonych klas, w których nowa klasa posiada atrybuty należące do innych klas.

\+ **Abstrakcja**

# Podstawowe podejścia: w skrócie

Uniwersytet Wrocławski

Naiwne:

- każdy rzeczownik w opisie projektu odpowiada klasie,
- każdy czasownik odpowiada metodzie.

Mądre:

- klasy służą do odseparowania zmiennych części projektu,
- skupienie na definicji interfejsów.

# Dobre zasady

Uniwersytet Wrocławski

1. Jasne definicje interfejsów.
2. Chowanie implementacji (ona może się zmieniać - interfejs powinien być stabilny).
3. Klasy pomagają w opanowaniu zmienności właśnie dzięki interfejsom i enkapsulacji.
4. Zasady SOLID:
   - S - Single-responsiblity principle
   - O - Open-closed principle
   - L - Liskov substitution principle
   - I - Interface segregation principle
   - D - Dependency Inversion Principle

Uniwersytet Wrocławski

# OOP w R

# W skrócie

Uniwersytet Wrocławski

- Obiektowość w R wygląda zwykle inaczej niż ta typowa (którą znacie z C++ czy Pythona).
- Podstawowy system obiektowy w R – S3 – skupia się na funkcjach: metody nie należą do obiektów, ale do funkcji generycznych (generic).
- Bardziej formalny system – S4 – pozwala na formalne definiowanie obiektów.
- Klasy tworzone w systemie R6 odpowiadają typowemu programowaniu obiektowemu.

# S3

Uniwersytet Wrocławski

- Sprawdzenie klasy obiektu: class(x)
- W S3 nie definiujemy klas – definiujemy metody.
- Klasę obiektu nadajemy poprzez class(x)<- lub structure(x, class = )
- Jeśli chcemy mieć konstruktor, musimy go napisać sami (i nie będzie on jedynym sposobem, w jaki może powstać obiekt tej klasy).
- Dziedziczenie: class(x) dłuższe niż 1.
- We własnym kodzie: metoda.klasa = function()
- W pakiecie: UseMethod + metoda.klasa()
- Dokumentacja: jak funkcji + możliwość połączenia kilku opisów w jednym pliku dokumentacji.
  [Można dodać @method metoda klasa, gdy w nazwie metody / klasy występuje kropka]

# S4

Uniwersytet Wrocławski

- Sprawdzenie klasy: is(x)
- Definicja klasy: setClass(nazwa, slots, contains, prototype).
  - slots: atrybuty
  - contains: klasy, po których dziedziczy nowa klasa
  - prototype: domyślne wartości atrybutów
- Nowy obiekt: new()
- Definicja metody: setGeneric + setMethod. (+signature)
- Uwaga: metoda może być wybierana na podstawie więcej niż jednego obiektu.
- Poprawność obiektu: setValidity
- Dokumentacja: @slot opisuje atrybuty klasy; dokumentacja funkcji – jak w S3,
- Jeśli klasy S4 / metody są w osobnych plikach - @include

# R6

Uniwersytet Wrocławski

- Sprawdzenie klasy: class()
- Konstrukcja klasy: R6Class(nazwa, lista), w liście należy podać atrybuty i metody klasy (publiczne).
- Metody mogą się odwoływać do obiektu przez $self, ale nie wymagają tego parametru (inaczej niż w Pythonie).
- Nowy obiekt: klasa$new() - tak też wywołuje się inne metody. Metody zwracają invisible(self).
- Specjalne metody: initialize(), print()
- Dziedziczenie: parametr inherit przy definiowaniu klasy. Odwołanie do klasy nadrzędnej: $super (jak w Pythonie)
- Dokumentacja: @field zamiast @slot, docstring jak w Pythonie.

# Podsumowanie

Uniwersytet Wrocławski

1. Klasy typu S3 są najbardziej naturalne w R i wystarczą w większości przypadków.
2. Klasy typu S4 są (niemal) konieczne w pracy w problemach bioinformatycznych / biostatystycznych, bo opiera się na nich Bioconductor.
3. System S4 jest podobny do S3, ale bardziej sformalizowany. Niestety jest też trudniejszy w zrozumieniu (wymaga poznania dokumentacji z różnych źródeł i eksperymentów).
4. System R6 jest bardziej klasycznym systemem obiektowym: metody należa do obiektów, a obiekty można bez problemu modyfikować w miejscu.

# Literatura

Uniwersytet Wrocławski


1. https://adv-r.hadley.nz/oo-tradeoffs.html i reszta rozdziału o programowaniu obiektowym.
2. Clean Code, Clean Architecture.
3. https://www.bioconductor.org/help/course-materials/2017/Zurich/S4-classes-and-methods.html