# *Kovarijanca i sedam OOPL-a*

Zlatko Sirotić, univ.spec.inf.
ISTRA TECH d.o.o., Pula

ISTRATECH

# ISTRA TECH d.o.o. & me

JavaCro'22 Autumn | Jesen

- ISTRA TECH je novo ime (od 2015.) poduzeća **Istra informatički inženjering**, osnovanog 1990. godine.
- Radim na informatičkim poslovima od 1984. godine.
- Oracle softverske alate (baza, Designer CASE, Forms 4GL, Reports, Java) koristim oko 25 godina.
- Objavljivao sam stručne radove na kongresima / konferencijama HrOUG, JavaCro, CASE, KOM, "Hotelska kuća", te u časopisima "Mreža", "InfoTrend" i "UT".
- Neka moja programska rješenja objavljivana su na web stranicama firmi Oracle i Quest.
- Od 2012. sam vanjski suradnik na Fakultetu informatike Pula.

ISTRATECH

# HrOUG i JavaCro & me

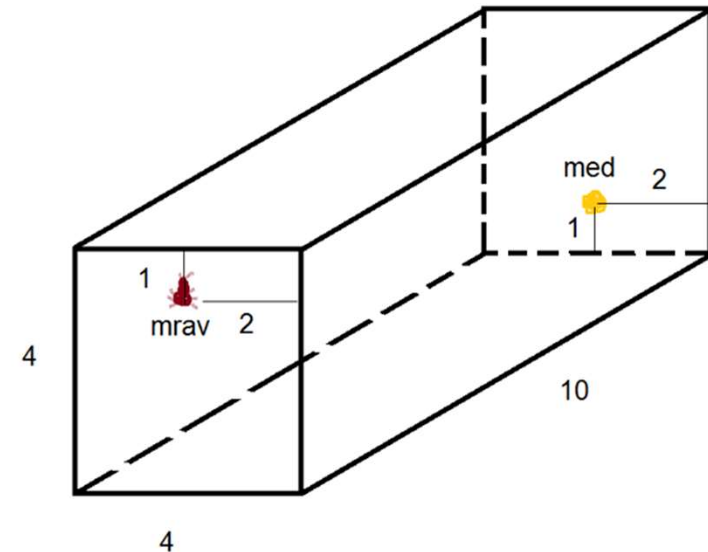
JavaCro'22 Autumn | Jesen

- Prvi put predavač na HrOUG 2002.
  Sudjelovao sam 18 puta i održao 24 predavanja.
- Prvi put predavač na JavaCro 2014.
  Sudjelovao sam 4 puta i održao 4 predavanja.
- "Odijelo ne čini čovjeka"
  i "Naslov ne čini prezentaciju", ali možda ipak pomaže:
  **2004.** Kako spriječiti "začarani krug"
  (rješavanje određenog tipa poslovnih pravila u Oracle BP)
  **2007.** Objektno-relacijske baze podataka – postoje li?
  **2013.** Što poslije Pascala? Pa … Scala!
  **2014.** Trebaju li nam distribuirane baze u vrijeme oblaka?
  **2015.** Višestruko nasljeđivanje - san ili Java 8?
  **2020.** Postoji li samo jedna "ispravna" arhitektura
  web poslovnih aplikacija

# Mrav i med na prizmi (i valjku)

- Na HrOUG 2015. prvi put sam spomenuo zadatak Mrav i med na prizmi (verzija 0), unutar predavanja Povratak u Prolog.

  Na HrOUG 2019. sam najavio prezentaciju (verzija 1).




- Na stranici http://www.istratech.hr/category/blog/

nalazi se najnovija verzija (2), uz još neka predavanja:

- Mrav i med na prizmi - verzija 2
- Povratak u Prolog - verzija 2
- Strukturna složenost algoritama

# Teme



- OOP jezici su (tiho, tiho) slavili 50 godišnjicu prije 5 godina!

- Kratko o povijesti programskih jezika (redom pojavljivanja):
  C++, Eiffel, Java, C#, Scala, Kotlin, Swift

- C++, Java, C#, Kotlin, Swift, Eiffel, Scala
  primjer paralelne hijerarhije klasa
  i pokušaj primjene kovarijance u nadjačanoj metodi

- Kako navodi Bertrand Meyer (tvorac jezika Eiffel) u knjizi OOSC2 (1997), općeniti primjeri s paralelnom hijerarhijom klasa mogli bi se rješavati primjenom **generičkih klasa** (Java generics, C++ templates). No Meyer smatra da to nije pravo rješenje, jer traži savršeno predviđanje kod modeliranja klasa.

# Klasa

- **Klasa**, a ne **objekt**, je osnovni element objektno-orijentiranih programskih jezika. Možda bi se trebali zvati COPL, umjesto OOPL ☺
- Klasa je dio programskog koda, ima osobine i modula i tipa. Pojednostavljeno se može reći da vrijedi formula:
  **KLASA = MODUL + TIP**
- Klasa definira podatke - **atribute** i ponašanje – **metode** (rutine, funkcije).
- Na temelju klase mogu se napraviti podklase, koje **nasljeđuje** (nad)klasu.
- Podklasa može imati nove atribute i metode, a može i **nadjačati** (overriding) nasljeđenu metodu.
- Jedan od "Algol-oidnih" jezika bio je Simula 1, koji je bio namijenjen uglavnom za simulacije. Kasnija verzija, **Simula 67** nastala je 1967. godine u Norveškoj, a autori su Kristen Nygaard i Ole-Johan Dahl - **prvi OOPL u povijesti!**

# C++ (kratka povijest)

- C (autor je Dennis Ritchie), također Algol-ov potomak, nastao je 1970. kao jezik za sistemsko programiranje operacijskog sustava UNIX. U isto vrijeme nastao je i Pascal, isto potomak Algol-a. Za većinu kasnijih programskih jezika možemo reći da (barem po sintaksi) pripadaju C ili Pascal "struji".
- Nadograđujući C sa objektno orijentiranim mogućnostima (uz zadržavanje kompatibilnosti), **Bjarne Stroustrup** je 1983. godine napravio C++. 1986. godine je objavio knjigu "The C++ Programming Language".
- Tokom vremena je C++ dobivao neke vrlo značajne mogućnosti, **koje na početku nije imao: višestruko nasljeđivanje, generičke klase (predloške), obradu iznimaka** (exceptions) i dr.
- 1997. godine donesen je ISO standard.
U standardu C++11 uvedene su npr. i lambda funkcije.
C++14, C++17 i C++20 donose dodatna poboljšanja.

# Eiffel (kratka povijest)

- Eiffel je 1985. godine dizajnirao (a 1986. je napravljen prvi compiler) **Bertrand Meyer**, jedan od autoriteta na području OOPL-a.
- Eiffel je od početka je podržavao **višestruko nasljeđivanje, generičke klase, obradu iznimaka, garbage collection i metodu Design by Contract (DBC).**
- Kasnije su mu dodani **agenti, nasljeđivanje implementacije** (uz nasljeđivanje tipa) i metoda za konkurentno programiranje Simple Concurrent Object-Oriented Programming (**SCOOP**).
- U široj je javnosti daleko manje poznat nego C++ i Java, ali ga mnogi autoriteti smatraju najboljim OOP jezikom. Eiffel je od 2005. godine ECMA standardiziran, a od 2006. ISO standardiziran.
- Zadnja verzija jezika je dio alata Eiffel Studio 22.05.

# Java (kratka povijest)

- Java 1.0 se pojavila 1996. i (u pravo vrijeme!) reklamirana je kao jezik za Internet, čime je odmah stekla ogromnu slavu.
- Ironija sudbine je da su Java apleti, koji su 1996. godine uzdigli Javu na pijedestal, danas mrtvi.
- Počeci Jave sežu u 1992., kada se zvala Oak i bila namijenjena za upravljanje uređajima za kabelsku televiziju i slične uređaje.
- **Sami autori su rekli da je Java = C++--**, tj. da je to pojednostavljeni (u pozitivnom smislu) C++.
- Nije stoga čudno da Java i C++ imaju sličnu sintaksu. Međutim, Java nije podskup C++ jezika.
- Također, iako jednostavniji nego C++, Java nije baš jednostavan jezik.
- Trenutačna verzija 19 je ne-LTS (Long-Term-Support). Ne-LTS verzije izlaze svakih 6 mjeseci. Zadnja LTS verzija je 17, prethodna je bila 11 (17-6), ali iduća će biti 21 (17+4).

# C# (kratka povijest)

- C# je 2000. godine dizajnirao **Anders Hejlsberg** iz Microsofta. Microsoft je tada predstavio C# zajedno s .NET platformom i Visual Studio alatom.
- U to vrijeme Microsoft nije imao proizvode otvorenog koda. Četiri godine kasnije, 2004., započeo je besplatni projekt otvorenog koda pod nazivom Mono, koji je pružao višeplatformski kompajler i okruženje za izvršavanje za programski jezik C#. Desetljeće kasnije, Microsoft je objavio Visual Studio Code (uređivač koda), Roslyn (kompajler) i unificiranu .NET platformu (softverski okvir), od kojih svi podržavaju C# i besplatni su, otvorenog koda i više platformi.
- C# je napravljen na temelju iskustava sa Javom (**npr. C# ima bolje riješene generičke klase nego Java**), ali ima i neke mogućnosti koje ima samo C++.
- C# je ubrzo postao međunarodni standard: ECMA (ECMA-334) 2002. i ISO (ISO/IEC 23270) 2003. Najnovija verzija jezika je C# 10.0, izašla 2021. godine.

# Scala (kratka povijest)

- Programski jezik Scala kreirao je **Martin Odersky**, profesor na Ecole Polytechnique Fédérale de Lausanne (EPFL).
- Krajem 80-ih doktorirao je na ETH Zürich kod profesora Niklausa Wirtha (kreatora Pascala i Module-2).
- Nakon toga naročito se **bavio istraživanjima u području funkcijskih jezika**, zajedno sa kolegom Philom Wadlerom (jednim od dva glavna kreatora funkcijskog jezika Haskell).
- Kada je izašla Java, Odersky i Wadler su 1996. napravili jezik **Pizza** nad JVM-om. Na temelju projekta Pizza, napravili su 1997./98. **Generic Java (GJ)**, koji je uveden u Javu 5.
- Odersky je 2002. počeo raditi novi jezik Scala. Tako je nazvana kako bi se naglasila njena **skalabilnost**.
- Prva javna verzija izašla je 2003. Trenutačna verzija je 3.1, (izašla 2021. godine), značajan skok u odnosu na verziju 2.

# Kotlin (kratka povijest)



- Programski jezik Kotlin predstavila je 2011. poznata firma JetBrains, a glavni autor je **Andrej Breslav**. Prva stabilna verzija 1.0 pojavila se 2016. JetBrains je "prepisao" izvorni kod svojih alata IntelliJ IDEA i Android Studio iz Jave u Kotlin.
- Kotlin je, kao i Scala, objektni jezik s funkcijskim proširenjima, koji radi (i) na JVM-u. Jezik je vrlo pragmatičan, tj. nije toliko napredan kao Scala, ali je lakši za učenje za Java programere. Moglo bi se reći da je Kotlin "bolja Java".
- Google je 2017. prihvatio Kotlin kao jedan od glavnih jezika za Android (uz Javu i C++). **U maju 2019., Google je izjavio da je Kotlin njihov preferirani jezik za Android programiranje.**
- **Kao i Scala, Kotlin ima**: eliminaciju repnog poziva (TCE); ime varijable piše se prije tipa varijable; varijable mogu biti imutabilne (**val**) i mutabilne (**var**); sintaksa kod funkcijskih proširenja je puno čitljivija od one u Javi. **Nema lijenu evaluaciju**, kao i dosta drugih naprednih mogućnosti.
- Trenutačna verzija je 1.7.10.

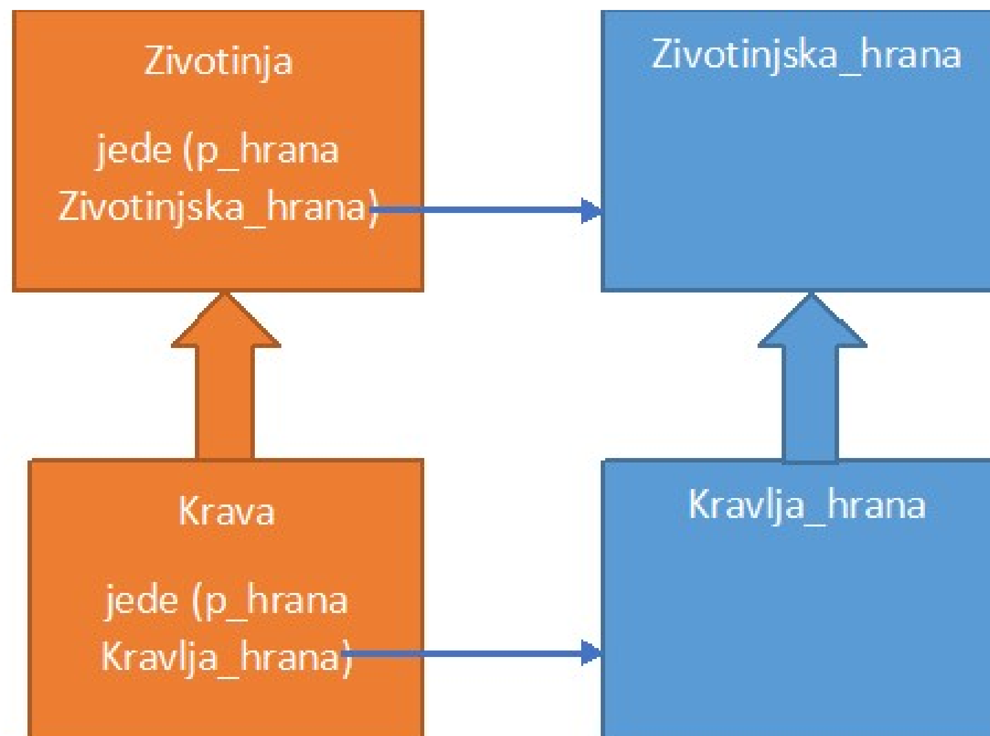
# Swift (kratka povijest)



- Programski jezik Swift su razvili Apple Inc. i zajednica otvorenog koda.
- Prvi put objavljen 2014., Swift je razvijen kao **zamjena za Appleov raniji programski jezik Objective-C**, koji je bio uglavnom nepromijenjen od ranih 1980-ih i nedostajale su mu značajke modernog jezika.
- Ključni aspekt Swiftovog dizajna bila je mogućnost interakcije s ogromnim tijelom postojećeg Objective-C koda razvijenog za Apple proizvode tijekom prethodnih desetljeća. Apple je želio da Swift podržava mnoge temeljne koncepte povezane s Objective-C, ali na "sigurniji" način.
- Prošao je nadogradnju na verziju 1.2 tijekom 2014. i veliku nadogradnju na Swift 2 na WWDC 2015. U početku je bio vlasnički jezik, a od verzije 2.2 je softver otvorenog koda pod Apache License.
- U prvom tromjesečju 2018. Swift je nadmašio Objective-C u popularnosti. Zadnja verzija je 5.7, od ove godine.

# Problem paralelne hijerarhije (povezanih) klasa

- Pretpostavimo da imamo klase **Zivotinja** i **Zivotinjska_hrana**, a klasa Zivotinja ima i proceduru **jede**, koja ima parametar tipa Zivotinjska_hrana.
- Sad želimo napraviti podklase **Krava** i **Kravlja_hrana** i u klasi Krava **promijeniti tip parametra** u proceduri jede iz Zivotinjska_hrana u Kravlja_hrana.



# Problem paralelne hijerarhije (povezanih) klasa



- To je **promjena signature u nadjačanoj metodi**.

- Općenito, kod OOP jezika postoje tri mogućnosti:

1. tip parametra se ne može mijenjati
   \- **bez varijance (no variance)**
2. tip parametra može se mijenjati tako da bude podtip u odnosu na bazni tip – **kovarijanca (covariance)**
3. tip parametra može se mijenjati tako da bude nadtip u odnosu na bazni tip - **kontravarijanca (contravariance)**.

- Dakle, mi bismo htjeli primijeniti **kovarijancu**.

ISTRATECH JavaCro'22 Autumn | Jesen

# C++ primjer paralelne hijerarhije

- Napravimo prvo apstraktne nadklase Zivotinjska_hrana i Zivotinja:

```
#include <string>
#include <iostream>
using namespace std;

class Zivotinjska_hrana {
public:
   string naziv;
   virtual string naziv_hrane() = 0; // abstract method
};

class Zivotinja {
public:
   string ime;
   virtual void jede(Zivotinjska_hrana* p_hrana) = 0;
};
```

# C++ primjer paralelne hijerarhije



- Primijetimo da u prethodnim klasama C++ traži ključnu riječ **virtual** za one metode koje u nastavku želimo nadjačati. U podklasama više nije nužno navoditi virtual, jer se to podrazumijeva, ali ga ovdje ipak eksplicitno pišemo.
- Zatim napravimo podklase od Zivotinjska hrana – prvo podklasa Kravlja_hrana.

```
class Kravlja_hrana : public Zivotinjska_hrana {
public:
  Kravlja_hrana(string p_naziv) {
    naziv = p_naziv;
  };

  virtual string naziv_hrane() override {
    return "Kravlja hrana: " + naziv;
  }
};
```

# C++ primjer paralelne hijerarhije



- Napomenimo da u C++ ključna riječ **override** postoji od C++11 (2011.).
- Zatim napravimo podklasu Riba (napomena: ovdje i u nastavku ćemo promatrati ribu isključivo kao vrstu hrane, iako je riba i životinja):

```
class Riba : public Zivotinjska_hrana {
public:
  Riba(string p_naziv) {
    naziv = p_naziv;
  };

  virtual string naziv_hrane() override {
    return "Riba: " + naziv;
  }
};
```

ISTRATECH

JavaCro'22 Autumn | Jesen

# C++ primjer paralelne hijerarhije

- Na kraju napravimo podklasu Krava, ali tako da u proceduri jede **ne pokušamo primijeniti kovarijancu** kod parametra p_hrana:

```
class Krava : public Zivotinja {
public:
   Krava(string p_ime) {
      ime = p_ime;
   };

   virtual void jede(Zivotinjska_hrana* p_hrana) override
   {
      cout << "Krava: " << ime <<
         " jede " << p_hrana->naziv_hrane() << "\n";
   }
};
```

# C++ primjer paralelne hijerarhije

- Sljedeći kod pokazuje da kravi možemo dati kravlju hranu (npr. travu), ali i ribu (npr. srdelu), a onda je samo korak do kravljeg ludila :)

```
int main() {
   Krava*          v_krava  = new Krava("Milka");
   Kravlja_hrana* v_khrana = new Kravlja_hrana("Trava");
   Riba*           v_riba   = new Riba("Srdela");

   v_krava->jede(v_khrana);
   v_krava->jede(v_riba);
}
```

```
Krava: Milka jede Kravlja hrana: Trava
Krava: Milka jede Riba: Srdela
```

# C++ primjer paralelne hijerarhije

- Naravno, nismo zadovoljni prethodnim ponašanjem, pa **pokušavamo primijeniti kovarijancu, što ne uspijeva:**

```
class Krava : public Zivotinja {
public:
  Krava(string p_ime) : Zivotinja(p_ime) {};
  virtual void jede(Kravlja_hrana* p_hrana) override {
    cout << "Krava: " << ime <<
      " jede " << p_hrana->naziv_hrane() << "\n";
  }
};
```

```
[Error] 'virtual void Krava::jede(Kravlja_hrana*)'
  marked override, but does not override
```

# Java primjer paralelne hijerarhije

- Napravimo prvo apstraktne nadklase Zivotinjska_hrana i Zivotinja:

```
abstract class Zivotinjska_hrana {
   String naziv;
   abstract String naziv_hrane();
}

abstract class Zivotinja {
   String ime;
   abstract void jede(Zivotinjska_hrana p_hrana);
}
```

ISTRATECH

JavaCro'22 Autumn | Jesen

# Java primjer paralelne hijerarhije

- Zatim napravimo podklase od Zivotinjska hrana – prvo podklasa Kravlja_hrana:
- Napomenimo da anotacija **@Override** postoji od Jave 5.

```
class Kravlja_hrana extends Zivotinjska_hrana {
  Kravlja_hrana(String p_naziv) {
    naziv = p_naziv;
  };

  @Override
  String naziv_hrane() {
    return "Kravlja hrana: " + naziv;
  }
}
```

ISTRATECH

JavaCro'22 Autumn | Jesen

# Java primjer paralelne hijerarhije

- Zatim napravimo podklasu Riba:

```
class Riba extends Zivotinjska_hrana {
  Riba(String p_naziv) {
    naziv = p_naziv;
  };

  @Override
  String naziv_hrane() {
    return "Riba: " + naziv;
  }
}
```

# Java primjer paralelne hijerarhije

- Na kraju napravimo podklasu Krava, ali tako da u proceduri jede **ne pokušamo primijeniti kovarijancu** kod parametra p_hrana:

```
class Krava extends Zivotinja {
  Krava(String p_ime) {
    ime = p_ime;
  };

  @Override
  void jede(Zivotinjska_hrana p_hrana) {
    System.out.println
      ("Krava: " + ime +
       " jede " + p_hrana.naziv_hrane());
  }
}
```

# Java primjer paralelne hijerarhije



- Dobijemo isto ponašanje kao u C++.
  Sljedeći kod pokazuje da kravi možemo dati kravlju hranu (npr. travu), ali i ribu (npr. srdelu):

```
public class Test {
  public static void main(String[] args) {
    Krava          v_krava  = new Krava("Milka");
    Kravlja_hrana v_khrana = new Kravlja_hrana("Trava");
    Riba           v_riba   = new Riba("Srdela");

    v_krava.jede(v_khrana);
    v_krava.jede(v_riba);
  }
}
```

```
Krava: Milka jede Kravlja hrana: Trava
Krava: Milka jede Riba: Srdela
```

# Java primjer paralelne hijerarhije



- Naravno, nismo zadovoljni prethodnim ponašanjem, pa **pokušavamo primijeniti kovarijancu, što ne uspijeva:**

```
class Krava extends Zivotinja {
  Krava(String p_ime) {
    ime = p_ime;
  };

  @Override
  void jede(Kravlja_hrana p_hrana) {
    System.out.println("Krava: " + ime +
      " jede " + p_hrana.naziv_hrane());
  }
}
Test.java:39: error: method does not override or
  implement a method from a supertype
  @Override
  ^
```

ISTRATECH

JavaCro'22 Autumn | Jesen

# C# primjer paralelne hijerarhije

- Napravimo prvo apstraktne nadklase Zivotinjska_hrana i Zivotinja:

```
abstract class Zivotinjska_hrana {
  public string naziv;
  public abstract string naziv_hrane();
}

abstract class Zivotinja {
  public string ime;
  public abstract void jede(Zivotinjska_hrana p_hrana);
}
```

# C# primjer paralelne hijerarhije



- Zatim napravimo podklase od Zivotinjska hrana:

```
class Kravlja_hrana : Zivotinjska_hrana {
  public Kravlja_hrana(string p_naziv) {naziv =
   p_naziv;}

  public override string naziv_hrane()
    {return "Kravlja hrana: " + naziv;}
}

class Riba : Zivotinjska_hrana {
  public Riba(string p_naziv) {naziv = p_naziv;}

  public override string naziv_hrane()
    {return "Riba: " + naziv;}
}
```

# C# primjer paralelne hijerarhije

- Na kraju napravimo podklasu Krava, ali tako da u proceduri jede **ne pokušamo primijeniti kovarijancu** kod parametra p_hrana:

```
class Krava : Zivotinja {
  public Krava(string p_ime) {
    ime = p_ime;
  }

  public override void jede(Zivotinjska_hrana p_hrana) {
    Console.WriteLine("Krava: " + ime +
      " jede " + p_hrana.naziv_hrane());
  }
}
```

# C# primjer paralelne hijerarhije

- Dobijemo isto ponašanje kao u C++ i Javi. Sljedeći kod pokazuje da kravi možemo dati kravlju hranu (npr. travu), ali i ribu (npr. srdelu):

```
public class Test {
  public static void Main(string[] args) {
    Krava          v_krava   = new Krava("Milka");
    Kravlja_hrana v_khrana = new Kravlja_hrana("Trava");
    Riba           v_riba    = new Riba("Srdela");

    v_krava.jede(v_khrana);
    v_krava.jede(v_riba);
  }
}
Krava: Milka jede Kravlja hrana: Trava
Krava: Milka jede Riba: Srdela
```

ISTRATECH

JavaCro'22 Autumn | Jesen

# C# primjer paralelne hijerarhije

- Naravno, nismo zadovoljni prethodnim ponašanjem, pa **pokušavamo primijeniti kovarijancu, što ne uspijeva:**

```
class Krava : Zivotinja {
  public Krava(string p_ime) {ime = p_ime;}

  public override void jede(Kravlja_hrana p_hrana) {
    Console.WriteLine("Krava: " + ime +
      " jede " + p_hrana.naziv_hrane());
  }
}
```

```
Compilation error ...: 'Krava.jede(Kravlja_hrana)': no
   suitable method found to override
Compilation error ...: 'Krava' does not implement
   inherited abstract member
   'Zivotinja.jede(Zivotinjska_hrana)'
```

ISTRATECH

JavaCro'22 Autumn | Jesen

# Kotlin primjer paralelne hijerarhije

- Napravimo prvo apstraktne nadklase Zivotinjska_hrana i Zivotinja:

```
private abstract class Zivotinjska_hrana(var naziv:
   String) {
   abstract fun naziv_hrane(): String
}

private abstract class Zivotinja (var ime: String) {
   abstract fun jede(p_hrana: Zivotinjska_hrana)
}
```

# Kotlin primjer paralelne hijerarhije

- Zatim napravimo podklase od Zivotinjska hrana:

```
private class Kravlja_hrana(naziv: String) :
   Zivotinjska_hrana(naziv) {

   override fun naziv_hrane(): String
      {return "Kravlja hrana: " + naziv}
}

private class Riba(naziv: String) :
   Zivotinjska_hrana(naziv) {

   override fun naziv_hrane(): String
      {return "Riba: " + naziv}
}
```

ISTRATECH

# Kotlin primjer paralelne hijerarhije

JavaCro'22 Autumn | Jesen

- Na kraju napravimo podklasu Krava, ali tako da u proceduri jede **ne pokušamo primijeniti kovarijancu** kod parametra p_hrana:

```
private class Krava(ime: String) : Zivotinja(ime) {

  override fun jede(p_hrana: Zivotinjska_hrana) {
    println("Krava: " + ime +
      " jede " + p_hrana.naziv_hrane())
  }
}
```

# Kotlin primjer paralelne hijerarhije

- Dobijemo isto ponašanje kao u C++, Javi i C#. Sljedeći kod pokazuje da kravi možemo dati kravlju hranu (npr. travu), ali i ribu (npr. srdelu):

```
fun main() {
   val v_krava:   Krava          = Krava("Milka")
   val v_khrana: Kravlja_hrana = Kravlja_hrana("Trava")
   val v_riba:    Riba           = Riba("Srdela")

   v_krava.jede(v_khrana)
   v_krava.jede(v_riba)
}
```

```
Krava: Milka jede Kravlja hrana: Trava
Krava: Milka jede Riba: Srdela
```

# Kotlin primjer paralelne hijerarhije

- Naravno, nismo zadovoljni prethodnim ponašanjem, pa **pokušavamo primijeniti kovarijancu, što ne uspijeva:**

```
private class Krava(ime: String) : Zivotinja(ime) {

  override fun jede(p_hrana: Kravlja_hrana) {
    println("Krava: " + ime +
      " jede " + p_hrana.naziv_hrane())
  }
}
```

```
Class 'Krava' is not abstract and does not implement
  abstract base class
member public abstract fun jede(p_hrana:
  Zivotinjska_hrana): Unit
defined in Zivotinja 'jede' overrides nothing
```

ISTRATECH JavaCro'22 Autumn | Jesen

# Swift primjer paralelne hijerarhije

- Napravimo prvo nadklase Zivotinjska_hrana i Zivotinja (neapstraktne, jer Swift nema apstraktne klase):

```
import Foundation
class Zivotinjska_hrana {
  var naziv: String
  init(p_naziv: String) {naziv = p_naziv}
  func naziv_hrane() -> String {
    fatalError("Subclasses need to implement
      the 'naziv_hrane' method.")
  }
}
class Zivotinja {
  var ime: String
  init(p_ime: String) {ime = p_ime}
  func jede(p_hrana: Zivotinjska_hrana) {
    fatalError("Subclasses need to implement
      the 'jede' method.")
  }
}
```

ISTRATECH

JavaCro'22 Autumn | Jesen

# Swift primjer paralelne hijerarhije

- Zatim napravimo podklase od Zivotinjska hrana:

```
class Kravlja_hrana: Zivotinjska_hrana {
  override func naziv_hrane() -> String {
    return "Kravlja hrana: " + naziv
  }
}

class Riba: Zivotinjska_hrana {
  override func naziv_hrane() -> String {
    return "Riba: " + naziv
  }
}
```

# Swift primjer paralelne hijerarhije



- Na kraju napravimo podklasu Krava, ali tako da u proceduri jede **ne pokušamo primijeniti kovarijancu** kod parametra p_hrana:

```
class Krava: Zivotinja {
  override func jede(p_hrana: Zivotinjska_hrana) {
    print("Krava: " + ime +
      " jede " + p_hrana.naziv_hrane())
  }
}
```

# Swift primjer paralelne hijerarhije



- Dobijemo isto ponašanje kao u C++, Javi. C# i Kotlinu. Sljedeći kod pokazuje da kravi možemo dati kravlju hranu (npr. travu), ali i ribu (npr. srdelu):

```
let v_krava:   Krava         = Krava(p_ime: "Milka")
let v_khrana: Kravlja_hrana = Kravlja_hrana
                                   (p_naziv: "Trava")
let v_riba:    Riba          = Riba(p_naziv: "Srdela")

v_krava.jede(p_hrana: v_khrana)
v_krava.jede(p_hrana: v_riba)
```

```
Krava: Milka jede Kravlja hrana: Trava
Krava: Milka jede Riba: Srdela
```

ISTRATECH JavaCro'22 Autumn | Jesen

# Swift primjer paralelne hijerarhije

- Naravno, nismo zadovoljni prethodnim ponašanjem, pa **pokušavamo primijeniti kovarijancu, što ne uspijeva:**

```
private class Krava : Zivotinja {
  override func jede(p_hrana: Kravlja_hrana) {
    print("Krava: " + ime +
      " jede " + p_hrana.naziv_hrane())
  }
}
```

```
...error: method does not override any method from its
  superclass
  override func jede(p_hrana: Kravlja_hrana) {
                 ^
...note: potential overridden instance method
  'jede(p_hrana:)' here
  func jede(p_hrana: Zivotinjska_hrana) {}
       ^
```

# Swift primjer paralelne hijerarhije



- Ako pokušamo primijeniti **protocol** (nešto kao Java interface), pokazuje se da on **ne omogućava kovarijancu**:

```
protocol Zivotinja_prot {
  associatedtype Hrana
  func jede(p_hrana: Hrana)
}

class Zivotinja: Zivotinja_prot {
  var ime: String
  init(p_ime: String) {ime = p_ime}
  func jede(p_hrana: Zivotinjska_hrana) {fatalError...}
}

class Krava: Zivotinja {
  // error: method does not override
  // any method from its superclass
  override func jede(p_hrana: Kravlja_hrana) {print...}
}
```

ISTRATECH

# Swift primjer paralelne hijerarhije

JavaCro'22 Autumn | Jesen

-  Niti typealias ne omogućava kovarijancu:

```
class Zivotinja {
  typealias Hrana = Zivotinjska_hrana
  var ime: String
  init(p_ime: String) {ime = p_ime}
  func jede(p_hrana: Hrana) {fatalError...}
}

class Krava: Zivotinja {
  typealias Hrana = Kravlja_hrana
  // error: 'Hrana' is ambigous
  // for type lookup in this context
  // found this candidate
  // typealias Hrana = Kravlja_hrana
  // typealias Hrana = Zivotinjska_hrana
  // error: method does not override
  // any method from its superclass
  override func jede(p_hrana: Hrana) {...}
}
```

# Eiffel primjer paralelne hijerarhije



- Za razliku od programskih jezika C++, Java, C#, Kotlin i Swift, koji kod nadjačavanja procedura ne dozvoljavaju kovarijancu kod parametara (dozvoljavaju kovarijancu samo kod povratne vrijednosti funkcije), Eiffel to dozvoljava.
- I ne samo kod parametara procedure:
  **Eiffel dozvoljava kovarijancu i kod nadjačanih atributa (u podklasama)**.

- No, kako ćemo vidjeti, Eiffelovo rješenje nije idealno.
- Bertrand Meyer je to jako lijepo opisao već u svojoj knjizi Object-Oriented Software Construction, release 2 (OOSC2) iz 1997. godine.

# Eiffel primjer paralelne hijerarhije

-  Napravimo prvo apstraktne (deferred) nadklase Zivotinjska_hrana i Zivotinja:

```
deferred class ZIVOTINJSKA_HRANA
feature
  naziv: STRING

  naziv_hrane: STRING deferred end
end

deferred class ZIVOTINJA
feature
  ime: STRING

  jede(p_hrana: ZIVOTINJSKA_HRANA) deferred end
end
```

ISTRATECH JavaCro'22 Autumn | Jesen

# Eiffel primjer paralelne hijerarhije

- Zatim napravimo podklase od Zivotinjska hrana – prvo podklasa Kravlja_hrana.

```
class KRAVLJA_HRANA inherit ZIVOTINJSKA_HRANA
create make

feature
  make (p_naziv: STRING)
  do
    naziv := p_naziv
  end

  naziv_hrane: STRING
  do
    Result := "Kravlja hrana: " + naziv
  end
end
```

ISTRATECH

# Eiffel primjer paralelne hijerarhije

JavaCro'22 Autumn | Jesen

- Zatim napravimo podklasu Riba:

```
class RIBA inherit ZIVOTINJSKA_HRANA
create make

feature
  make (p_naziv: STRING)
  do
    naziv := p_naziv
  end

  naziv_hrane: STRING
  do
    Result := "Riba: " + naziv
  end
end
```

# Eiffel primjer paralelne hijerarhije

- Na kraju napravimo podklasu Krava, ali tako da u proceduri jede **primijenimo kovarijancu** kod parametra p_hrana:

```
class KRAVA inherit ZIVOTINJA
create make

feature
   make (p_ime: STRING)
   do
      ime := p_ime
   end

   jede(p_hrana: KRAVLJA_HRANA)
   do
      print ("Krava: " + ime +
         " jede " + p_hrana.naziv_hrane + "%N")
   end
end
```

# Eiffel primjer paralelne hijerarhije



-  Sljedeći kod (prvi dio procedure make) pokazuje da kravi NE možemo direktno dati ribu (npr. srdelu) - kompajler se buni:

```
class APPLICATION
create make
feature
  make
  local
    v_krava:   KRAVA
    v_khrana:  KRAVLJA_HRANA
    v_riba:    RIBA
  do
    create v_krava.make ("Milka")
    create v_khrana.make ("Trava")
    create v_riba.make ("Srdela")
    v_krava.jede(v_khrana)
  --v_krava.jede(v_riba) -- greška kod kompajliranja ...
```

ISTRATECH JavaCro'22 Autumn | Jesen

# Eiffel primjer paralelne hijerarhije

- Nažalost, **kravi ipak možemo dati ribu indirektno, pomoću polimorfne dodjele**.
- Istina, runtime javi grešku, ali tada je možda već prekasno:

```
...   print ("- Polimorfna dodjela -" + "%N")
      v_zivotinja := v_krava -- polimorfna dodjela
      v_zivotinja.jede(v_khrana)
      v_zivotinja.jede(v_riba) -- runtime poruka!
  end
end
```

```
Krava: Milka jede Kravlja hrana: Trava
- Polimorfna dodjela -
Krava: Milka jede Kravlja hrana: Trava
Catcall detected in {KRAVA}.jede for arg#1: expected
  KRAVLJA_HRANA but got RIBA
Krava: Milka jede Riba: Srdela
```

# Eiffel primjer paralelne hijerarhije

- Kako navodi Bertrand Meyer u knjizi OOSC2, općeniti primjeri s paralelnom hijerarhijom klasa mogli bi se rješavati primjenom **generičkih klasa** (Java generics, C++ templates). No Meyer smatra da je to nije pravo rješenje, jer traži savršeno predviđanje kod modeliranja klasa.
- U OOSC2 Meyer daje tri moguća rješenja problema koji je naveden u prethodnom primjeru. Jedno rješenje je **globalna analiza sustava**, gdje bi se analizirao (tj. kompajlirao) cijeli sustav, a ne samo klase koje su mijenjane. Jer, programski kod može biti ispravan na razini pojedinačnih klasa, a neispravan na razini sustava. Naravno, kompajliranje svih klasa nakon promjene u jednoj klasi nije baš jako praktično.
- Na stranicama firme **eiffel.com** (vlasnik je Meyer) našli smo informaciju za EiffelStudio 14.05 (May 2014): **Compiler Covariance fix: new mechanism to avoid catcall at compile time** – ali, nema detalja.

# Scala varijanca generičkih parametara



- **Scala ne dozvoljava varijancu parametara kod nadjačavanja procedura**, nego ima **varijancu kod generičkih parametara u generičkim klasama**, što ima i Kotlin, a Java nema.
- Kod Scale, deklaracija npr. **List[+A]** označava **kovarijancu** (**kontravarijanca** se označava s **List[-A]**), što znači da je npr. **List[String] podtip od List[AnyRef]** - podudara se s time što je **String podtip od AnyRef** (kod kontravarijance, List[String] je nadtip od List[AnyRef]).
- Takav način deklariranja varijance generičkih parametara u generičkim klasama zove se **declaration-site variance.** Java ima samo **use-site variance**, gdje se varijanca deklarira tek kod korištenja generičke klase, u metodama**.**
- To znači da u Javi moguće probleme oko varijance uvijek rješava korisnik library-a, a u Scali / Kotlinu ponekad može rješavati pisac library-a (ne uvijek, jer i u Scali / Kotlinu ponekad treba use-site variance).

# Scala apstraktni tipovi

- Scala, uz apstraktne metode, ima i **apstraktne tipove** i apstraktne atribute (val i var):

```
abstract class Abstract {
  type T
  def transform(x: T): T
  val initial: T
  var current: T
}

class Concrete extends Abstract {
  type T = String
  def transform(x: String) = x + x
  val initial = "hi"
  var current = initial
}
```

# Scala primjer paralelne hijerarhije

- Prvo napravimo apstraktnu klasu Zivotinjska_hrana sa apstraktnom metodom naziv_hrane (pa u podklasama ne trebamo pisati override) i podklase Kravlja_hrana i Riba:

```
abstract class Zivotinjska_hrana(naziv: String) {
  def naziv_hrane: String
}

class Kravlja_hrana(var naziv: String)
extends Zivotinjska_hrana(naziv) {
  override def naziv_hrane: String =
    "Kravlja hrana: " + naziv
}

class Riba(var naziv: String)
extends Zivotinjska_hrana(naziv) {
  override def naziv_hrane: String =
    "Riba: " + naziv
}
```

# Scala primjer paralelne hijerarhije



- Onda napravimo apstraktnu klasu Zivotinja sa **apstraktnim tipom HRANA, ali ograničenim na tip Zivotinjska_hrana**.
- Na kraju i njenu podklasu Krava, tako da apstraktni tip HRANA zamijenimo sa konkretnim tipom Zivotinjska_hrana:

```
abstract class Zivotinja(ime: String) {
  type HRANA <: Zivotinjska_hrana // <: označava podtip

  def jede(p_hrana: HRANA): Unit
}

class Krava(var ime: String) extends Zivotinja(ime) {
  type HRANA = Kravlja_hrana

  override def jede(p_hrana: Kravlja_hrana) =//ili HRANA
    println
      ("Krava: " + ime + " jede " + p_hrana.naziv_hrane)
}
```

# Scala primjer paralelne hijerarhije

- Kompajler ne pušta da kravi damo ribu, niti direktno, niti nakon polimorfne dodjele:

```
object Test {
  def main(args: Array[String]): Unit = {
    val v_krava  = Krava("Milka")
    val v_khrana = Kravlja_hrana("Trava")
    val v_riba   = Riba("Srdela")
    v_krava.jede(v_khrana)
    v_krava.jede(v_riba) // greška kod kompajliranja
    val v_zivotinja = v_krava; // polimorfna dodjela
    v_zivotinja.jede(v_khrana)
    v_zivotinja.jede(v_riba) // greška kod kompajliranja
  }
}
... error: type mismatch;
 found    : Riba
 required: Kravlja_hrana
    v_krava.jede(v_riba) i v_zivotinja.jede(v_riba)
                 ^
```

# Scala apstraktni tipovi : generičke klase

- Dakle, iako Scala ne dozvoljava kovarijancu parametara kod nadjačavanja procedura, ona ipak rješava problem paralelnih hijerarhija, pomoću apstraktnih tipova.

- Što kažu u knjizi **Programming Scala** (Dean Wampler i Alex Payne, 2. izdanje iz 2014., str. 368-369):

***Technically, you could implement almost all the idioms that parameterized types support using abstract types and vice versa.*** *However, in practice, each feature is a natural fit for different design problems.*

***Parameterized types work nicely for containers, like collections,*** *where there is little connection between the types represented by the type parameter and the container itself. For example, a list works the same if it's a list of strings, a list of doubles, or a list of integers…*

***In contrast, abstract types tend to be most useful for type "families", types that are closely linked.***

# Što kaže krava

Za type klučnu reč,
Kaj dala si mi, Scala,
Kaj morem ti neg' reć:
Od vsega srca fala.

ISTRATECH

JavaCro'22 Autumn | Jesen

# Je li je Scala type zaista idealno rješenje?

- Kako smo već naveli, Bertrand Meyer u OOSC2 navodi da bi se problemi s paralelnom hijerarhijom klasa mogli rješavati pomoću generičkih klasa, ali problem je što one traže savršeno predviđanje kod modeliranja klasa.
- **Ali, zar to nije slučaj i kad se koristi Scala type** (iako je rješenje jednostavnije od generičkih klasa) **- u nadklasi moramo predvidjeti potrebu za type?**

```
abstract class Zivotinja(ime: String) {
  type HRANA <: Zivotinjska_hrana
  def jede(p_hrana: HRANA)
}
class Krava(var ime: String)
 extends Zivotinja(ime) {
  type HRANA = Kravlja_hrana
  override def jede(p_hrana: HRANA) = ...
}
```