TECHIN

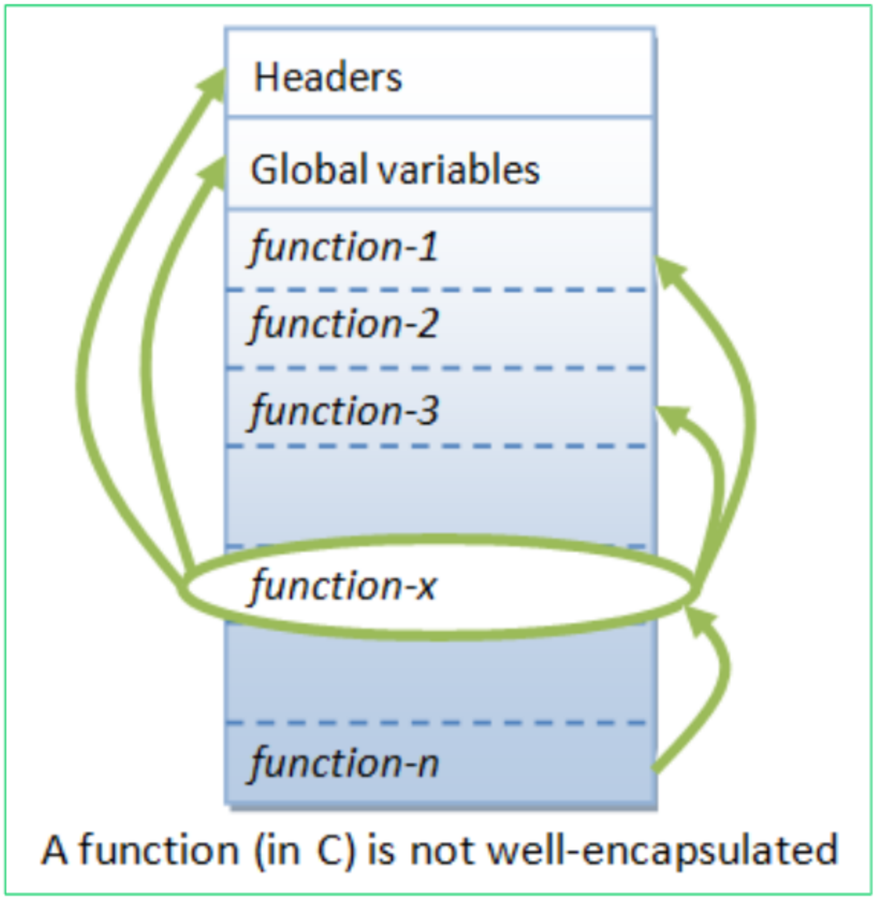
08 Objektinis programavimas (I)

Jaroslav Grablevski

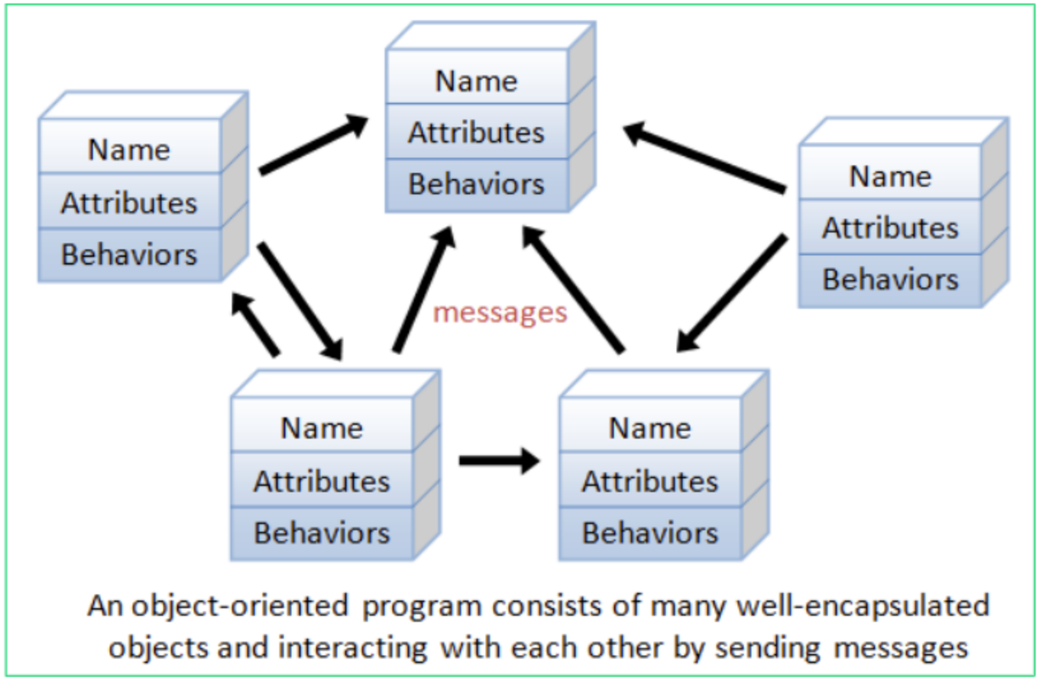
Turinys

- Pagrindinės sąvokos
- Objektas
- Klasė
- Kintamieji
- Metodai
- Paketai
- Konstruktoriai
- Inkapsuliacija
- toString()

Procedural vs. Object-Oriented



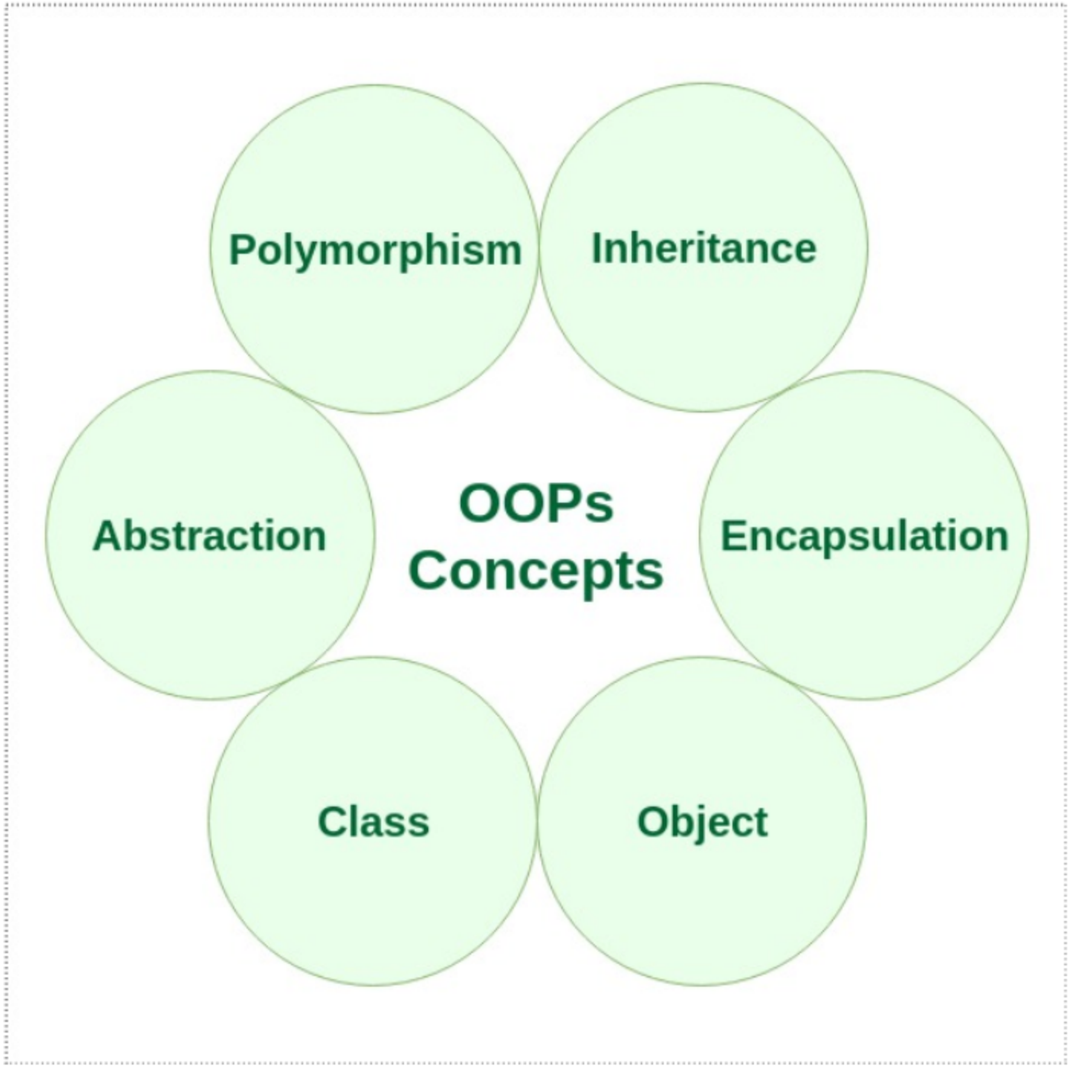
A function (in C) is not well-encapsulated



An object-oriented program consists of many well-encapsulated objects and interacting with each other by sending messages

Pagrindinės OOP sąvokos ir terminai

- Klasė
- Objektas
- Abstrakcija
- Inkapsuliacija
- Paveldėjimas
- Polimorfizmas



Abstrakcija (abstraction)

- Tai yra apibendrinimo principas, kuris nusako, kad konkrečius dalykus galime apibendrinti, išskiriant esmines ir pačias svarbiausias jų savybes bei atsiribojant nuo specifinių detalių.
- Tai yra viena iš pagrindinių priemonių kovojant su programinės įrangos sudėtingumu.
- Abstraktus duomenų tipas - programuotojo sukurtas duomenų tipas, kuriame nustatomas duomenų reikšmių vaizdavimas ir kartu apibrėžiamos galimos šiam tipui operacijos.

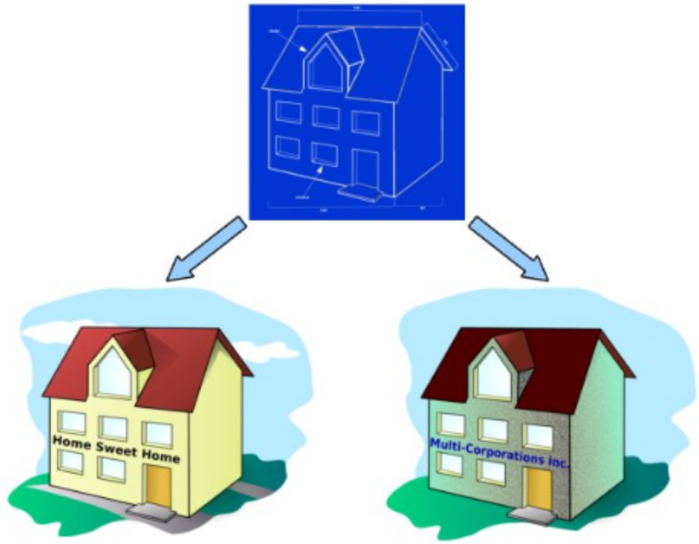
Objektas

Tai savarankiškas programinis komponentas turintis šias savybes:

- Tapatybė (identity (or name))
- Būsena (state or attributes, variables, fields)
- Elgesys (behavior or methods)

Klasė

- Klasė – tai objekto šablonas (brėžinys), kurį sudaro duomenys (kintamieji, laukai) ir metodai.
- Iš vienos klasės galima sukurti kiek norima objektų, tačiau kiekvienas objektas yra sukurtas iš vienos konkrečios klasės.



Klasė

- Klasės kūnas susideda iš tokių elementų:
  - Objekto kintamųjų
  - Konstantų
  - Konstruktorių
  - Metodų
  - ...

***Nė vienas nėra privalomas**

```
public class Nothing {
    // there is nothing
}
```

Klasė

```
class Circle {
  /** The radius of this circle */
  double radius = 1;

  /** Construct a circle object */
  Circle() {
  }

  /** Construct a circle object */
  Circle(double newRadius) {
    radius = newRadius;
  }

  /** Return the area of this circle */
  double getArea() {
    return radius * radius * Math.PI;
  }

  /** Return the perimeter of this circle */
  double getPerimeter() {
    return 2 * radius * Math.PI;
  }

  /** Set a new radius for this circle */
  void setRadius(double newRadius) {
    radius = newRadius;
  }
}
```

Data field

Constructors

Method

Klasės/Objektai

Objekto kintamieji (instance variables, fields)

- Deklaruojami klasės ribose (bet ne metoduose, konstruktoriuose ar blokuose)
- Vadinami objekto laukais *(fields)*
- Priklauso konkrečiam klasės egzemplioriui/objektui (kiekvienas objektas turi nuosavas šių kintamųjų kopijas)
- Atmintis šiems kintamiesiems išskiriama kuriant objektą
- Prieinami:
  - Metoduose;
  - Konstruktoriuose;
  - Blokuose;
- Galima naudoti prieinamumo modifikatorius
- Priskiriamos numatytosios reikšmės
- Pasiekiami naudojant: <objektoNuoroda>.kintamasis
- Objekto kintamieji nėra pasiekiami statiniams metodams

Objekto kintamieji - numatytosios reikšmės

```
//compile error, because the local
//variables x and y are not initialized
public static void main(String[] args) {
    int x; // x has no default value
    String y; // y has no default value
    System.out.println("x is " + x);
    System.out.println("y is " + y);
}
```

```
class Student {
    String name; // default value null
    int age; // default value 0
    boolean isScienceMajor; // default value false
    char gender; // default value '\u0000'
}
```

Objekto kintamieji (instance variables)

```java
public class TV {
    int channel = 1;         //  \
    int volumeLevel = 1;     //   |-data fields
    boolean on = false;      //  /

    public TV() { // constructor
    }
    public void turnOn() { // turn on TV
        on = true;
    }
    public void turnOff() { // turn off TV
        on = false;
    }
    public void setChannel(int newChannel) {
        if (on && newChannel >= 1 && newChannel <= 120)
            channel = newChannel;
    }
    public void channelUp() {
        if (on && channel < 120)
            channel++;
    }
    //.....
```

```java
public static void main(String[] args) {
    TV tv1 = new TV();
    tv1.turnOn();
    tv1.channel = 20;
    tv1.setChannel(30);
    tv1.setVolume(3);

    TV tv2 = new TV();
    tv2.turnOn();
    tv2.channelUp();
    tv2.volumeUp();
}
```

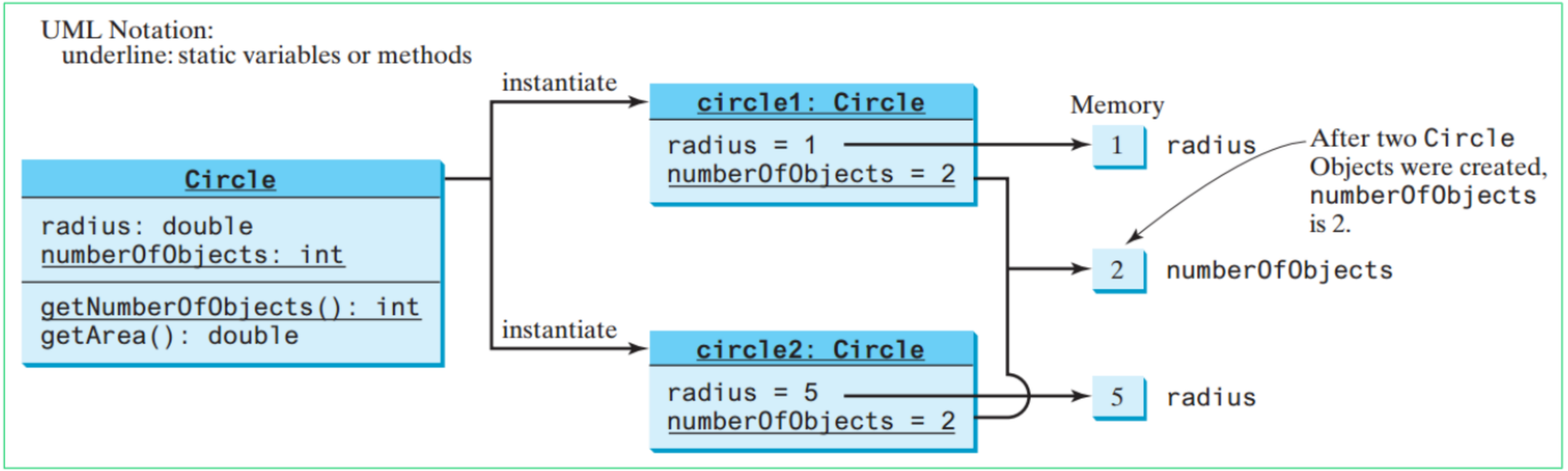
Klasės kintamieji (static class members)

- Deklaruojami klasės ribose (bet ne metoduose, konstruktoriuose ar blokuose)
- Priklauso konkrečiai klasei (viena kopija kintamųjų vienai klasei, nepriklausomai nuo to, kiek sukurta klasės objektų)
- Atmintis šiems kintamiesiems išskiriama programos startavimo metu
- Pasiekiami naudojant: <KlasėsPavadinimas>.kintamasis
- Dažniausiai naudojami kaip konstantos, prieinamos ir kitiems objektams

Klasės kintamieji (static class members)

```
// Static variable used to maintain a count of the number of
// Employee objects in memory.

public class Employee {
    private static int count = 0; // number of Employees created
    private String firstName;
    private String lastName;

    // initialize Employee, add 1 to static count and
    // output String indicating that constructor was called
    public Employee(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;

        ++count; // increment static count of employees
    }

    // static method to get static count value
    public static int getCount() {
        return count;
    }

}
```

Klasės kintamieji (static class members)



Konstantos

- Java programavimo kalboje konstanta yra realizuojama naudojant žodelį final, kuris nusako, kad reikšmė, kuri yra priskirta kintamajam yra nekintama (t.y. priskirtos reikšmės pakeisti nebegalima).

```
//klasės konstantos
public static final float PI = 3.14f;
public static final double G = 6.674 * Math.pow(10,-11);
public static final int C = 299_792_458;
public static final int ZERO = 0;

//objekto konstantos
private final String name;
private final int radius;
```

Konstruktorius

- Konstruktoriai skirti klasės inicializavimui - tai yra klasės būsenos paruošimui tuomet, kai sukuriamas objektas.
- Klasės konstruktoriai yra pagrindinė priemonė klasės objektams kurti.
- Panašus į metodą
- Vardas sutampa su klasės
- Neturi grąžinamo tipo (net void)
- Gali būti keli (0..*)

Konstruktorius

```
class Patient {

    String name;
    int age;
    float height;

    // konstruktorius
    public Patient(String name, int age, float height) {
        this.name = name;
        this.age = age;
        this.height = height;
    }
}
```

```
Patient patient1 = new Patient("Jonas", 40, 182.0f);
Patient patient2 = new Patient("Justina", 33, 171.5f);
```

Konstruktorius

- Klasė gali turėti daug konstruktorių, kurie gali būti perkrauti (overloaded)
- Būtina sąlyga – konstruktoriai turi skirtis savo antraštėmis, t.y. parametrų skaičiumi arba parametrų tipais.

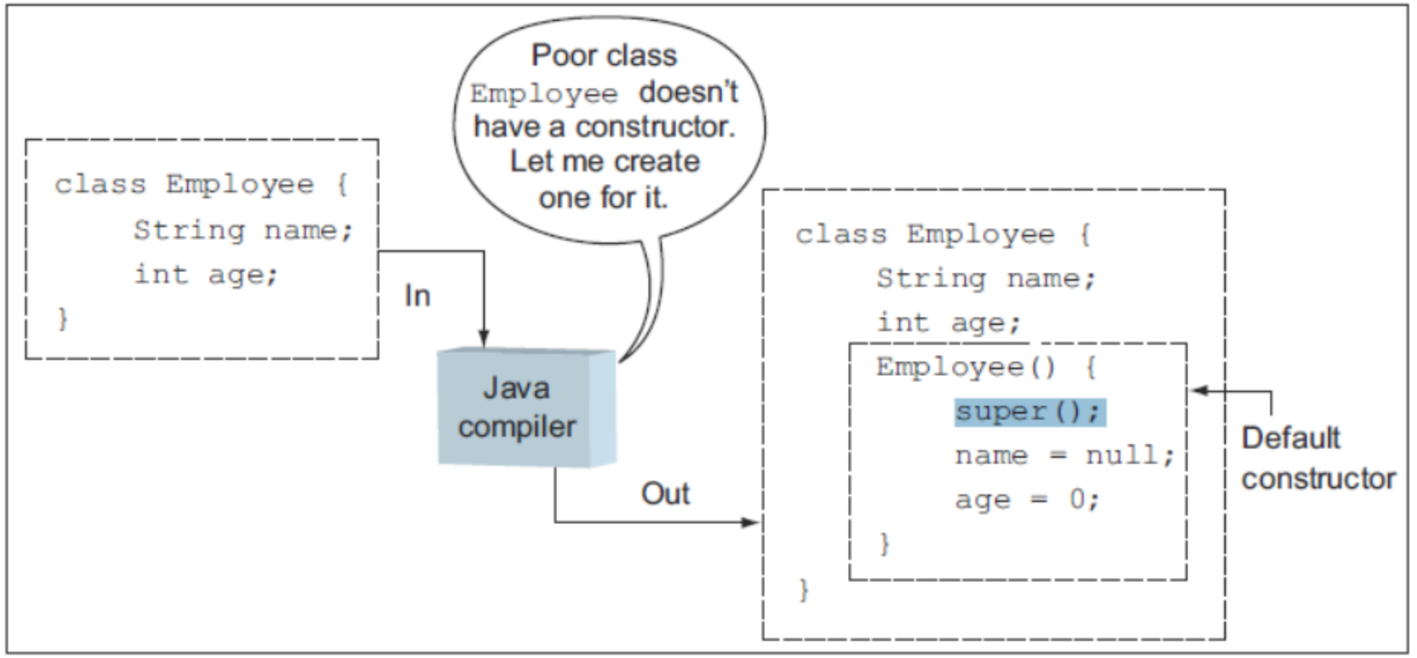
Konstruktorius

```java
public class Robot {
    String name;
    String model;

    public Robot() {
        this.name = "Anonymous";
        this.model = "Unknown";
    }

    public Robot(String name, String model) {
        this.name = name;
        this.model = model;
    }
}
```

```java
Robot anonymous = new Robot();
Robot andrew = new Robot("Andrew", "NDR-114");
```

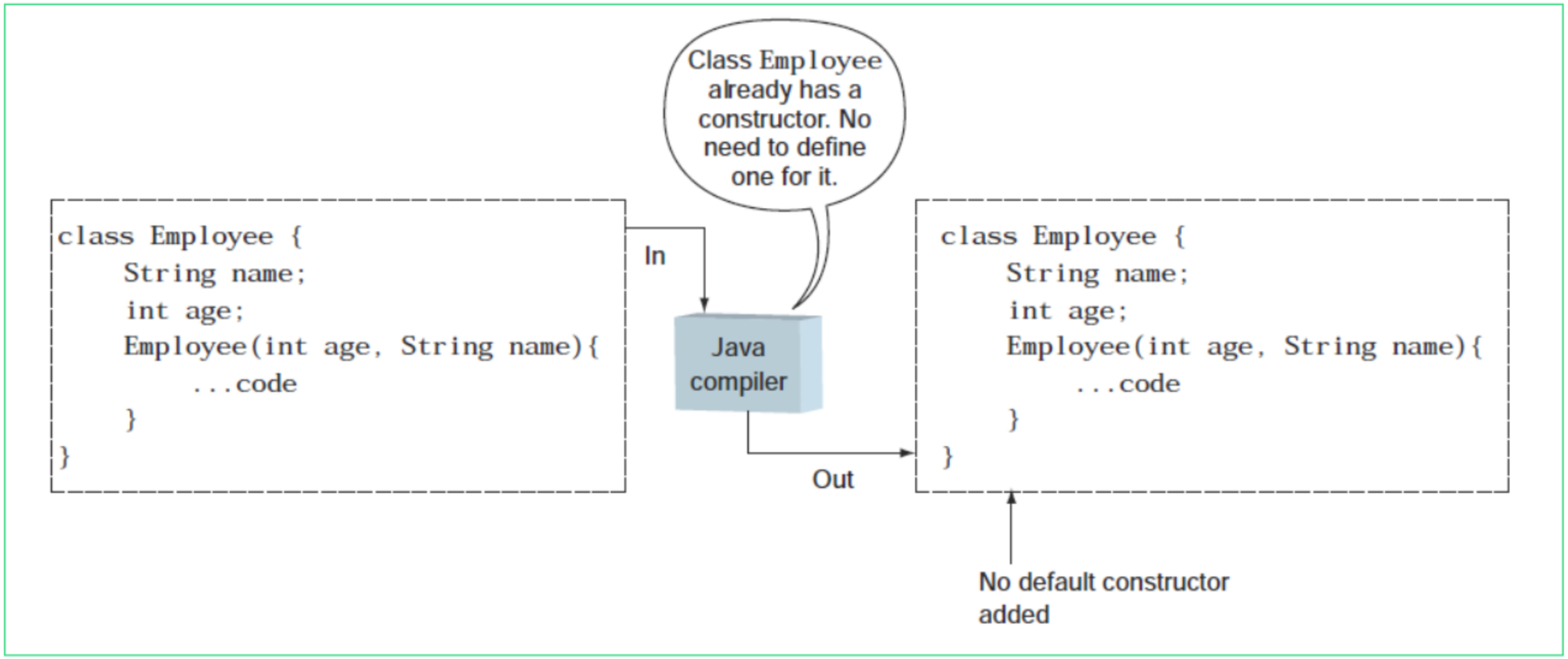
Konstruktorius (default constructor)

- Klasėje gali ir nebūti deklaruoto konstruktoriaus - tokiu atveju Java kompiliatorius sukurs numatytąjį konstruktorių - konstruktorius be parametrų.
- Jis visiems klasės kintamiesiems (laukams) suteikia ,,nulines“ reikšmes (pagal tipą).



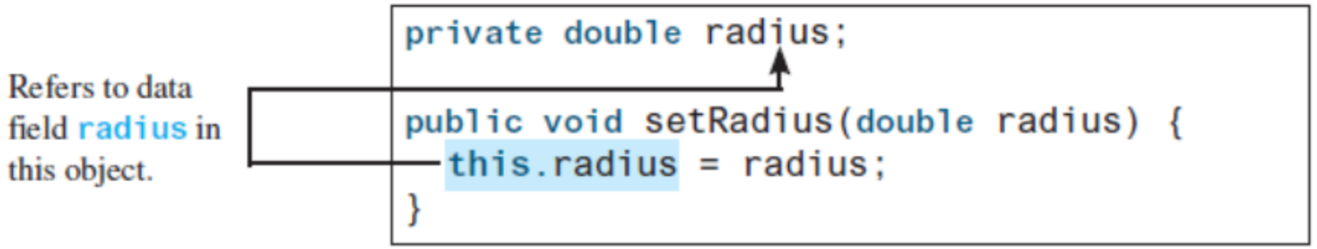
Konstruktorius (default constructor)

- Jei egzistuoja bent vienas deklaruotas konstruktorius, tuomet numatytasis konstruktorius nebus sukurtas.

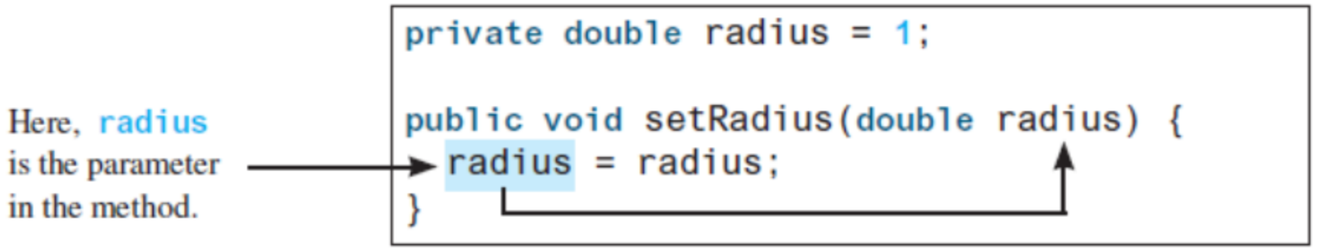


this

- Žodelis this, tai nuorodą į dabartinį objektą - nuoroda objekto, kurio metodas ar konstruktorius konkrečiu metu yra vykdomas.
- Dažniausiai naudojamas tais atvejais, kai metodo ar konstruktoriaus parametrų pavadinimai sutampa su objekto kintamųjų vardais.
- Papildomai, jis gali būti naudojamas tose vietose, kur reikalinga dabartinio objekto nuoroda.



(a) this.radius refers the radius data field in this object.



Konstruktorius > this

- Sakinys *this([<<argumentai>>])* yra naudojamas klasės konstruktoriuose iškviesti kitą toje pačioje klasėje esantį konstruktorių.
- Šis sakinys turi būti pats pirmasis konstruktoriuje.

```
public class Circle {
  private double radius;
  public Circle(double radius) {
    this.radius = radius;
  }

  public Circle() {
    this(1.0);
  }

  ...
}
```

The `this` keyword is used to reference the data field radius of the object being constructed.

The `this` keyword is used to invoke another constructor.

Kaip sukurti objektą?

- A constructor is invoked to create an object using the new operator.
- The variable myCircle holds a reference to a Circle object.

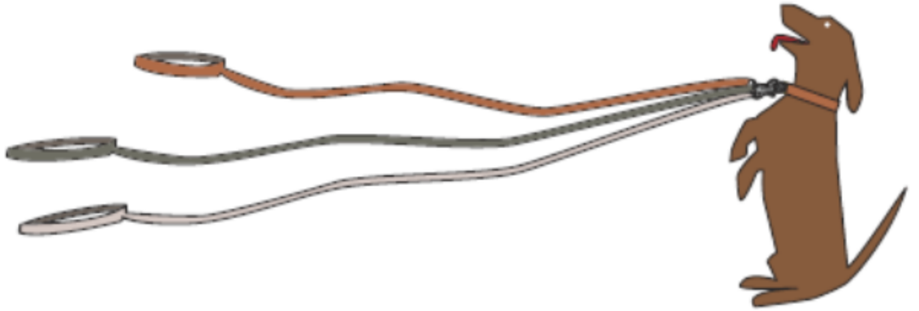
```
Circle myCircle = new Circle();
```

Reference Types

```
Dog d;
new Dog();
Dog dog = new Dog();
d = dog;
```

A leash without a dog.

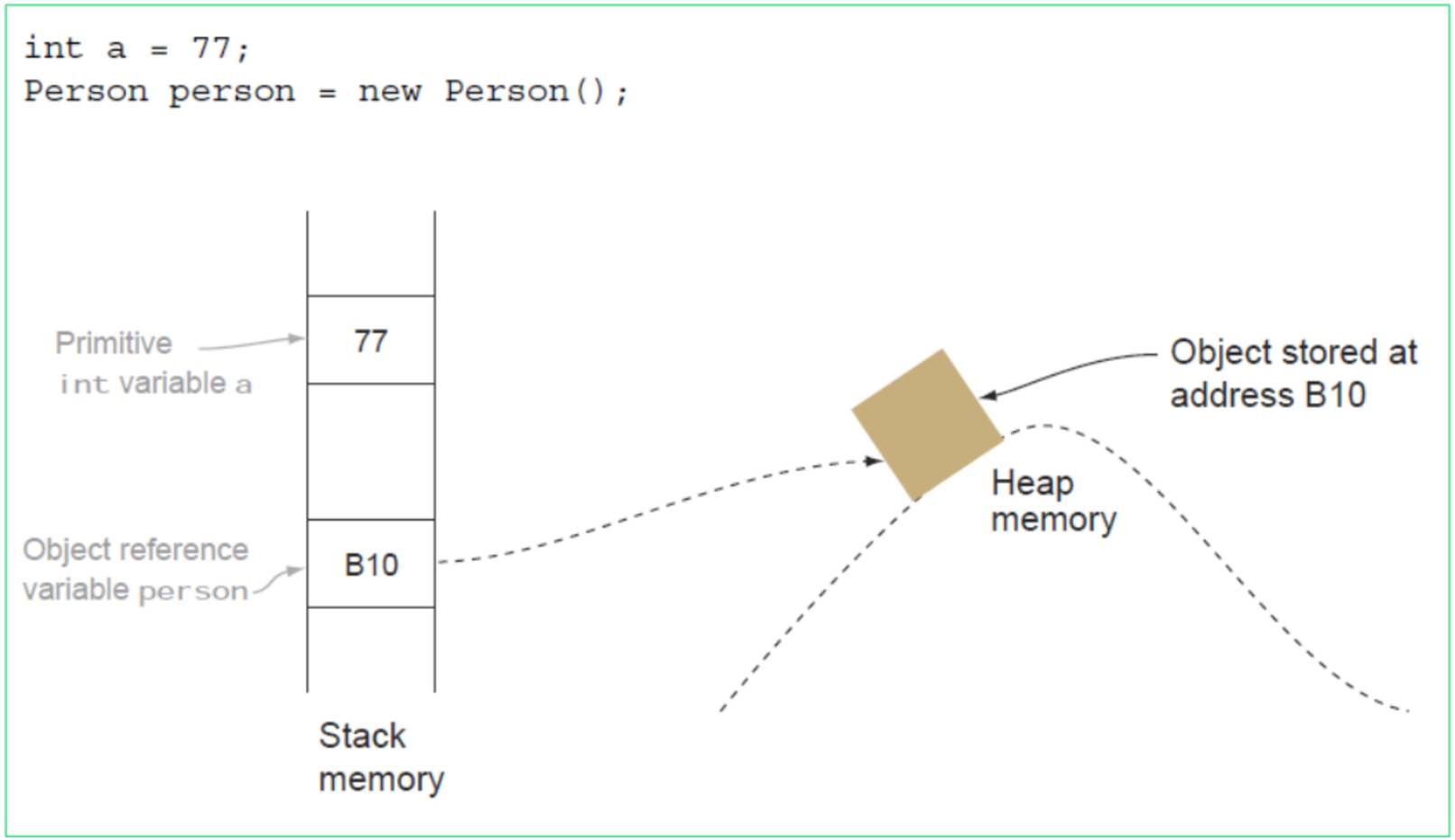
A dog without a leash.



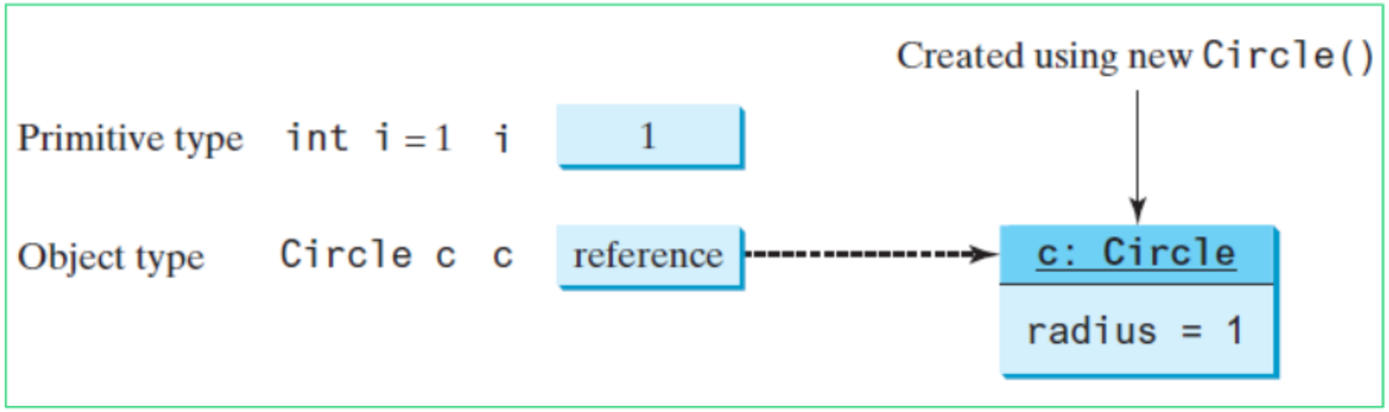
Several leashes may be tethered to one dog.

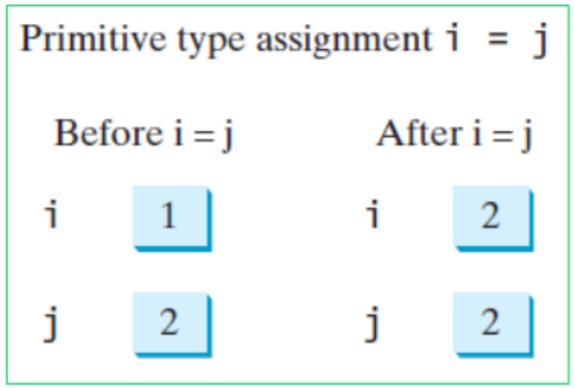
Reference Types

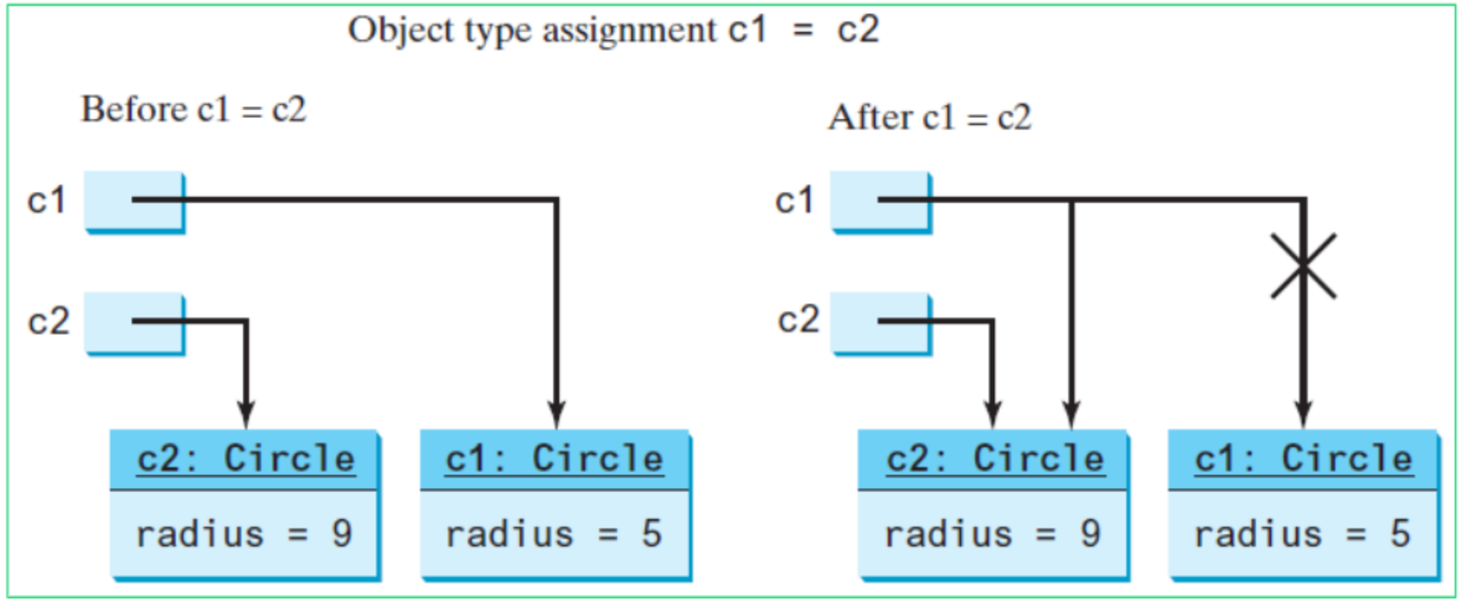
```
int a = 77;
Person person = new Person();
```



Variables of Primitive Types and Reference Types



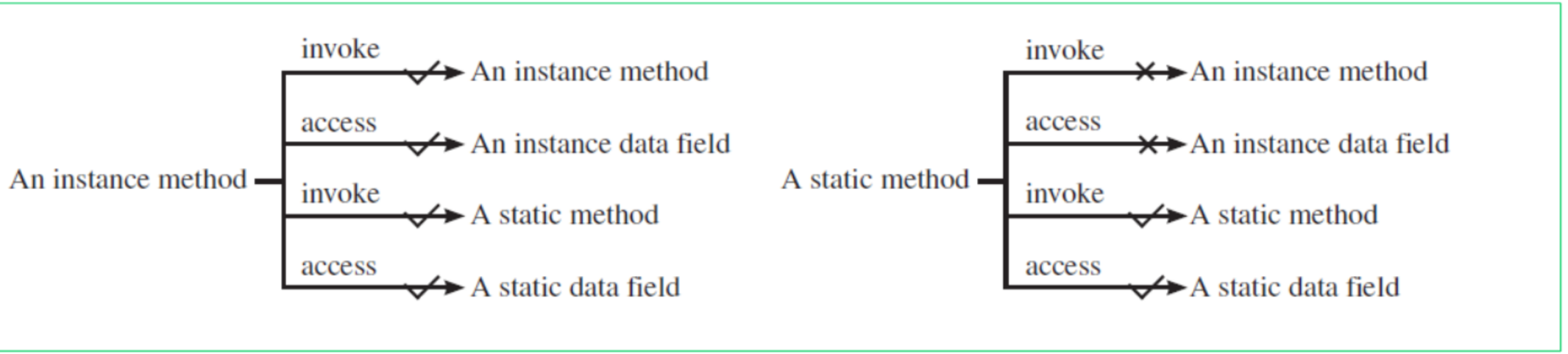




Metodai

- Metodai gali būti dvieju tipų:
  - Klasės (statiniai) metodai
    - Priklauso klasei
    - Norint iškviesti, nebūtinas objekto sukūrimas - užtenka klasės pavadinimo: *<klasėsPavadinimas>.<metodoPavadinimas>([<<argumentai>>])* (pvz. Math.pow(3,2))
    - Gali dirbti tik su klasės (statinius) kintamaisiais
    - Statiniai metodai pažymimi žodeliu static
  - Objekto metodai
    - Priklauso objektui
    - Norint iškviesti, būtinas objekto sukūrimas
    - Gali dirbti tiek su klasės, tiek su objekto kintamaisiais

Instance and static methods



Instance and static methods

```
public class Pvz {

    int i = 5;
    static int k = 2;

    public static void main(String[] args) {
        int j = i; // Wrong because i is an instance variable
        m1(); // Wrong because m1() is an instance method
    }
    public void m1() {
        //instance and static variables and methods
        // can be used in an instance method
        i = i + k + m2(i, k);
    }
    public static int m2(int i, int j) {
        return (int) (Math.pow(i, j));
    }
}
```

Metodai (overloading)

- Metodai gali būti perkrauti, t.y. klasėje gali būti deklaruoti daugiau nei vienas metodas su tuo pačiu pavadinimu.
- Svarbu, kad būtų tenkinamos šios taisyklės:
  - Parametrų kiekis turi skirtis
  - Turi skirtis parametrų tipai

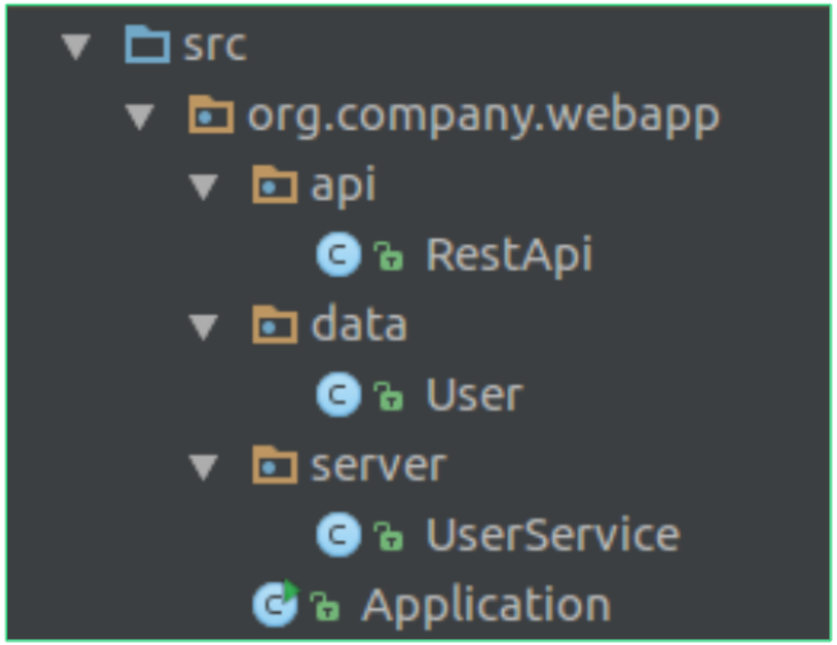
```
class Person{
    String name;
    int age;

    void speak(){

    void speak(int times){

    void speak(String lastName){

    void speak(int times, String lastName){

}
```

Paketai (packages)

- Paketas - tai vardų erdvė (angl. namespace), kurioje laikoma aibė susijusių klasių ir interfeisų. Paketai leidžia organizuoti programinius komponentus, panašiai, kaip katalogai leidžia organizuoti failus asmeniniame kompiuteryje.
- Papildomai, paketas leidžia slėpti tam tikras detales nuo kituose paketuose esančių klasių (inkapsuliacija), tokias kaip:
  - Klasės
  - Interfeisai
  - Kintamieji
  - Metodai

Packages

- In general, packages give us the following advantages:
  - to group related classes together that makes easier to figure out where a certain class you are looking for is;
  - to avoid the conflict of class names;
  - to control access to classes and members together with access modifiers (it's considered in another topic).
- To avoid creating packages with the same names as other public packages it is recommended that you start your package hierarchy with the reverse domain name of your company (or another organization).



Packages (import)

- If one of these classes is located in another package, to use one class inside another you should write an import statement using the keyword import.

```
import java.util.Scanner;
```

- It's possible to use a class from another package without performing import statement.

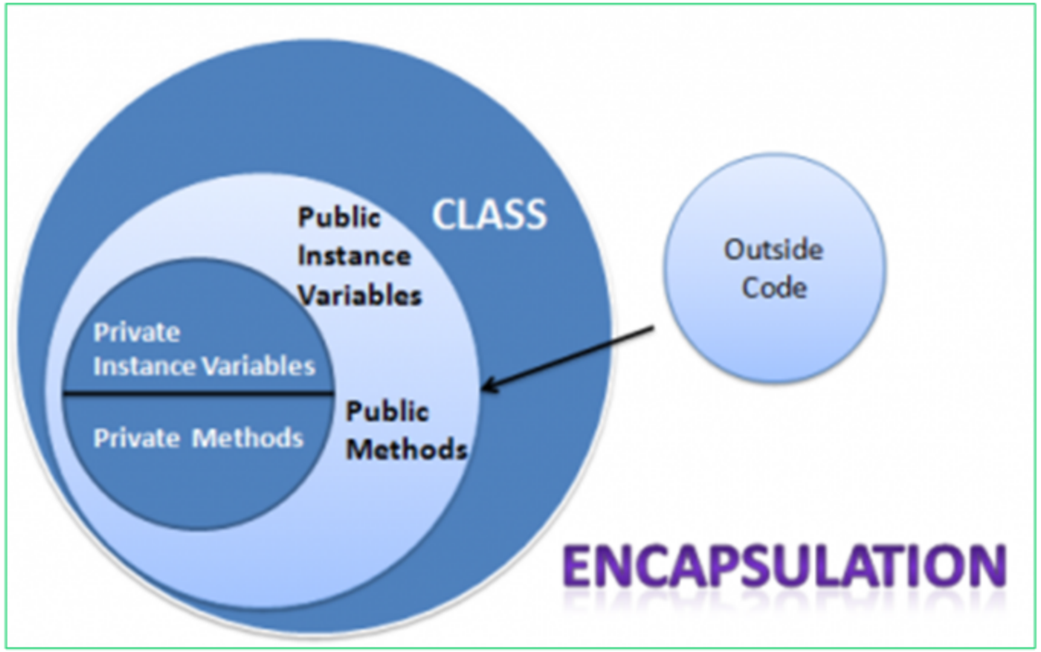
```
java.util.Scanner scanner = new java.util.Scanner(System.in);
```

Packages (static import)

```java
package it.akademija.java.oop;

//instead of the statement "import java.util.Arrays;"
import static java.util.Arrays.*;

public class Oop {

    public static void main(String[] args) {
        int[] numbers = { 1, 4, 5, 7, 5, 12 }; // an array

        sort(numbers); // instead of writing Arrays.sort(...)

        // instead of writing Arrays.copyOf(...)
        int[] copy = copyOf(numbers, numbers.length);
    }
}
```

Inkapsuliacija (encapsulation)

- Principas, pagal kurį objekto vidiniai duomenys yra slepiami ir jais galima manipuliuoti tik naudojant objekto viešus metodus.
- Programuotojas renkasi ką rodyti, o ką slėpti nuo objekto naudotojo (kito programuotojo).



Modifikatoriai / access modifiers

- Kintamųjų ir metodų galiojimo zoną papildomai reguliuoja modifikatoriai.

Modifier	Same class or nested class	Other class inside the same package	Extended Class inside another package	Non-extended inside another package
private	yes	no	no	no
default (package private)	yes	yes	no	no
protected	yes	yes	yes	no
public	yes	yes	yes	yes

Modifikatoriai / access modifiers

```
package  p1;

public class C1 {
  public int x;
  int y;
  private int z;

  public void m1() {
  }
  void m2() {
  }
  private void m3() {
  }
}
```

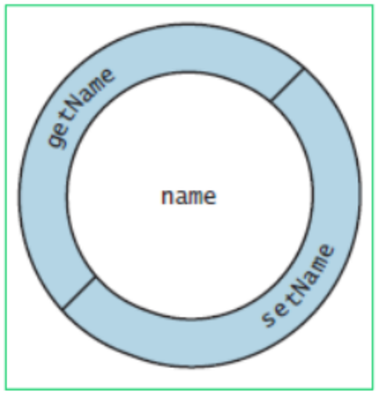
```
package  p1;

public class C2 {
  void aMethod() {
    C1 c1 = new C1();
    can access c1.x;
    can access c1.y;
    cannot access o.z;

    can invoke c1.m1();
    can invoke c1.m2();
    cannot invoke c1.m3();
  }
}
```

```
package p2;

public class C3 {
  void aMethod() {
    C1 c1 = new C1();
    can access c1.x;
    cannot access c1.y;
    cannot access c1.z;

    can invoke c1.m1();
    cannot invoke c1.m2();
    cannot invoke c1.m3();
  }
}
```

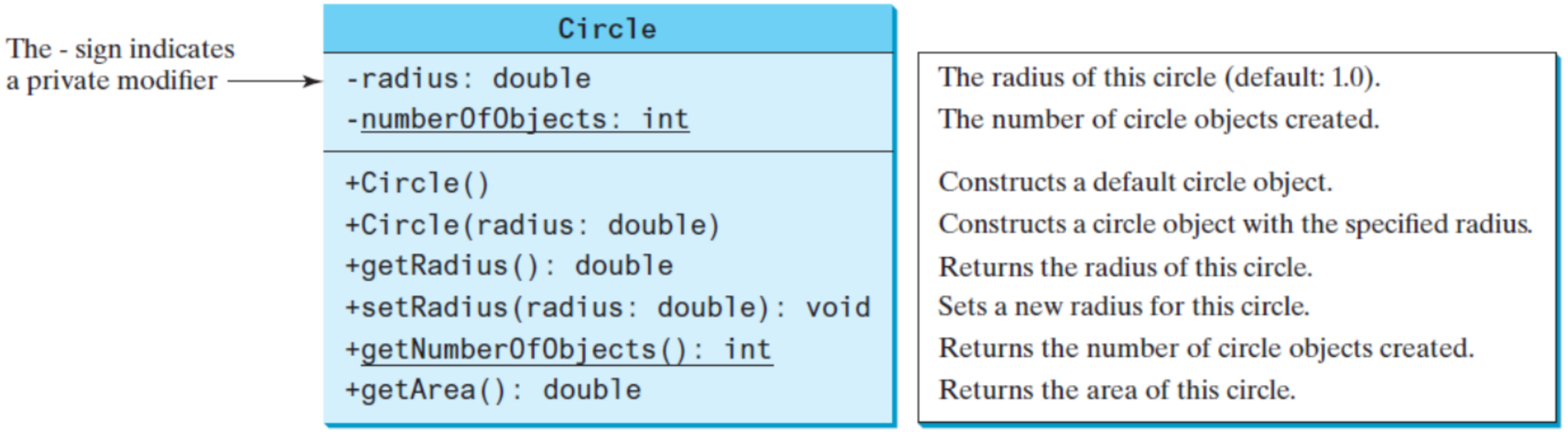
Getters / setters (accesors/mutators)

- To make a private data field accessible, provide a getter method to return its value.
- To enable a private data field to be updated, provide a setter method to set a new value.

```
//getter method
public returnType getPropertyName() {
//getter method if the returnType is boolean
public boolean isPropertyName() {
//setter method
public void setPropertyName(dataType propertyValue) {
```



Modifikatoriai / access modifiers



Modifikatoriai / access modifiers

```java
public class Circle {
    private double radius = 1; // encapsulate radius
    private static int numberOfObjects = 0;

    public Circle() {
        numberOfObjects++;
    }
    public Circle(double newRadius) {
        radius = newRadius;
        numberOfObjects++;
    }
    public double getRadius() {
        return radius;
    }
    public void setRadius(double newRadius) {
        radius = (newRadius >= 0) ? newRadius : 0;
    }
    public static int getNumberOfObjects() {
        return numberOfObjects;
    }
}
```

```java
public static void main(String[] args) {
    // Create a circle with radius 5.0
    Circle myCircle = new Circle(5.0);
    myCircle.radius = 5.5; //compile error
    // Increase myCircle's radius by 10%
    myCircle.setRadius(myCircle.getRadius() * 1.1);
    System.out.println("The radius of the circle is "
    + myCircle.getRadius());
    System.out.println("The number of objects created is "
    + Circle.getNumberOfObjects());
}
```

Metodas toString()

```
class Person{

}

public class ExampletoString {

    public static void main(String[] args) {
        Person p1 = new Person();
        System.out.println(p1);
    }
}
```

```
Person@15db9742
```

Metodas toString()

```java
class Person{

    private String name;
    private int age;

    public Person(String name, int age){
        this.name = name;
        this.age = age;
    }

    @Override
    public String toString() {
        return name + " is " + age + " years old.";
    }

}

public class ExampletoString {

    public static void main(String[] args) {
        Person p1 = new Person("Wiliam", 31);
        System.out.println(p1);
    }
}
```

```
Wiliam is 31 years old.
```