ПОДАТОЧНИ СТРУКТУРИ И АНАЛИЗА НА АЛГОРИТМИ

ЧАС 2: ВОВЕД ВО JAVA (ПРОДОЛЖЕНИЕ)

АУДИТОРИСКИ ВЕЖБИ

БОЈАНА ВЕЛИЧКОВСКА

НАСЛЕДУВАЊЕ

- Се користи клучниот збор **extends**.

```
public class superclass {  //основна класа
  . . .
  }
public class subclass extends superclass {  //изведена класа
  . . .
  }
```

- Наследувањето дозволува програмите да ги моделираат врските кои постојат во вистинскиот свет
- Повторно искористување на код

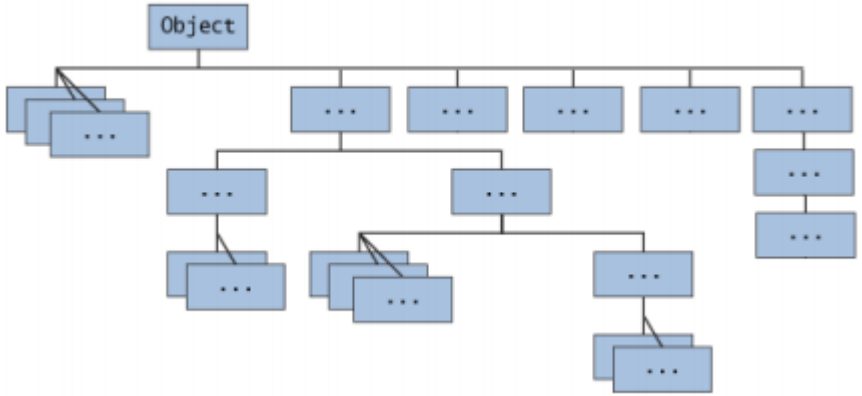
ХИЕРАРХИСКО (ПОВЕЌЕНИВОВСКО) НАСЛЕДУВАЊЕ

public class superclass{ . . . }

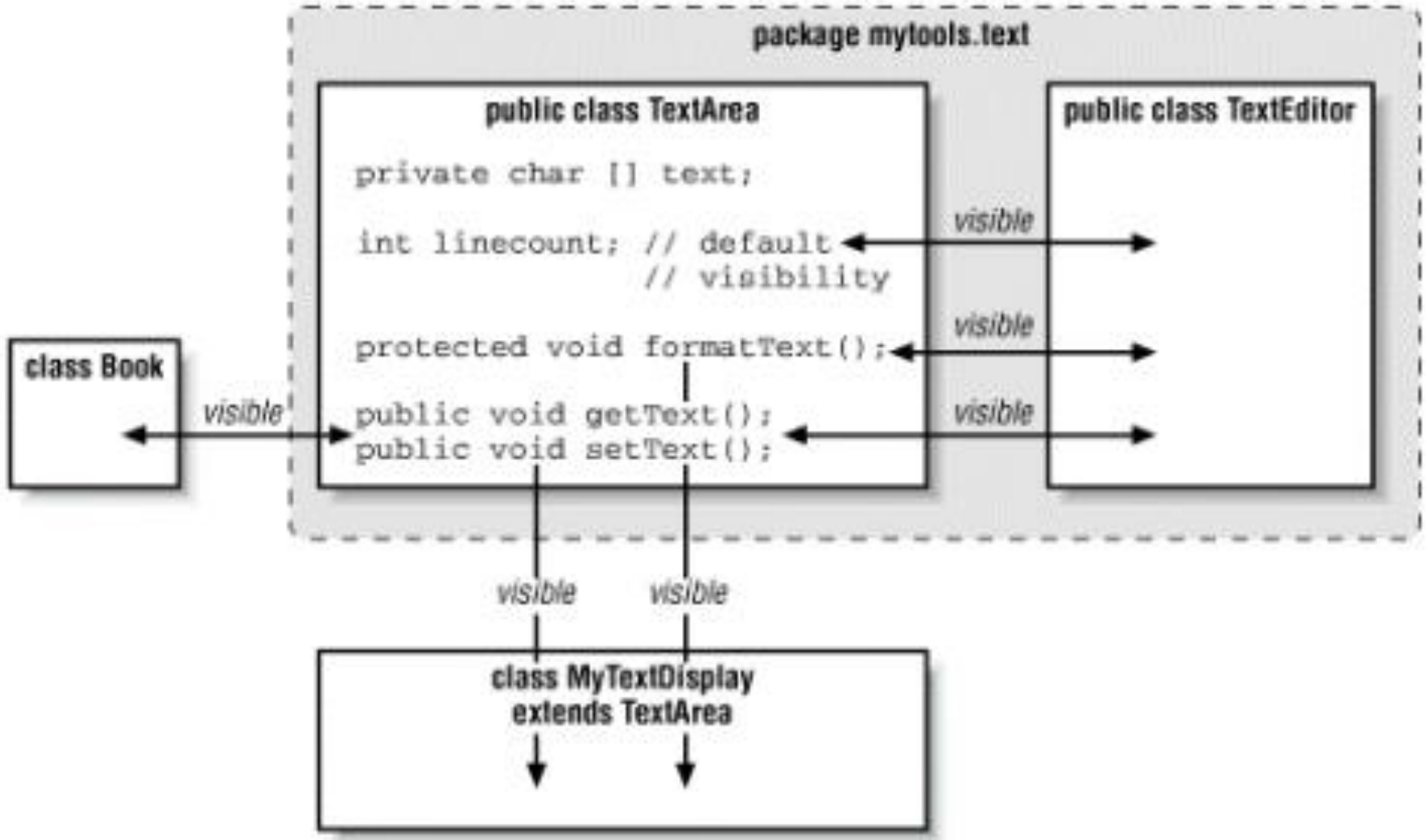
public class midclass extends superclass{ . . . }

public class childclass extends midclass{ . . . }

- Во Java не постои повеќекратно наследување
  - една класа директно наследува **само од една** класа
- Во отсуство на суперкласа, се претпоставува дека секоја класа наследува од класата Object



АТРИБУТИ ЗА ПРИСТАП



КОНСТРУКТОРИ ПРИ НАСЛЕДУВАЊЕ

- Повикување на конструктор од основната класа – super

```
public class Bicycle {

    // the Bicycle class has three fields
    public int cadence;
    public int gear;
    public int speed;

    // the Bicycle class has one constructor
    public Bicycle(int startCadence, int startSpeed, int startGear) {
        gear = startGear;
        cadence = startCadence;
        speed = startSpeed;
    }
```

```
public class MountainBike extends Bicycle {

    // the MountainBike subclass adds one field
    public int seatHeight;

    // the MountainBike subclass has one constructor
    public MountainBike(int startHeight, int startCadence,
                        int startSpeed, int startGear) {
        //calling the constructor of the superclass.
        super(startCadence, startSpeed, startGear);

        seatHeight = startHeight;
    }
```

SUPER

- Освен конструктор од основната класа, со super се повикува и метод од основната класа

```
class Animal {
  void eat( Food f ) {
// consume food
    }  }
class Herbivore extends Animal {
  void eat( Food f ) {
     // check if edible
     ...
     super.eat( f );
     }   }
```

SUPER

```
class superClass { . . . }
class midClass extends superClass { . . . }
class childClass extends midClass { . . . }
public static void main(String []args) {   superClass sp = new superClass();
  sp = new midClass();
  sp = new childClass();
  midClass mp;
  mp = new midClass();
  mp = new childClass();
}
```

INSTANCEOF

- Проверува чија инстанца е даден објект

```
class Parent {
  public Parent() {

  }
}
class Child extends Parent {
  public Child() {
    super();
  }
}
public class MainClass {
  public static void main(String[] a) {

    Child child = new Child();
    if (child instanceof Parent) {
      System.out.println("true");
    }
  }
}
```

FINAL

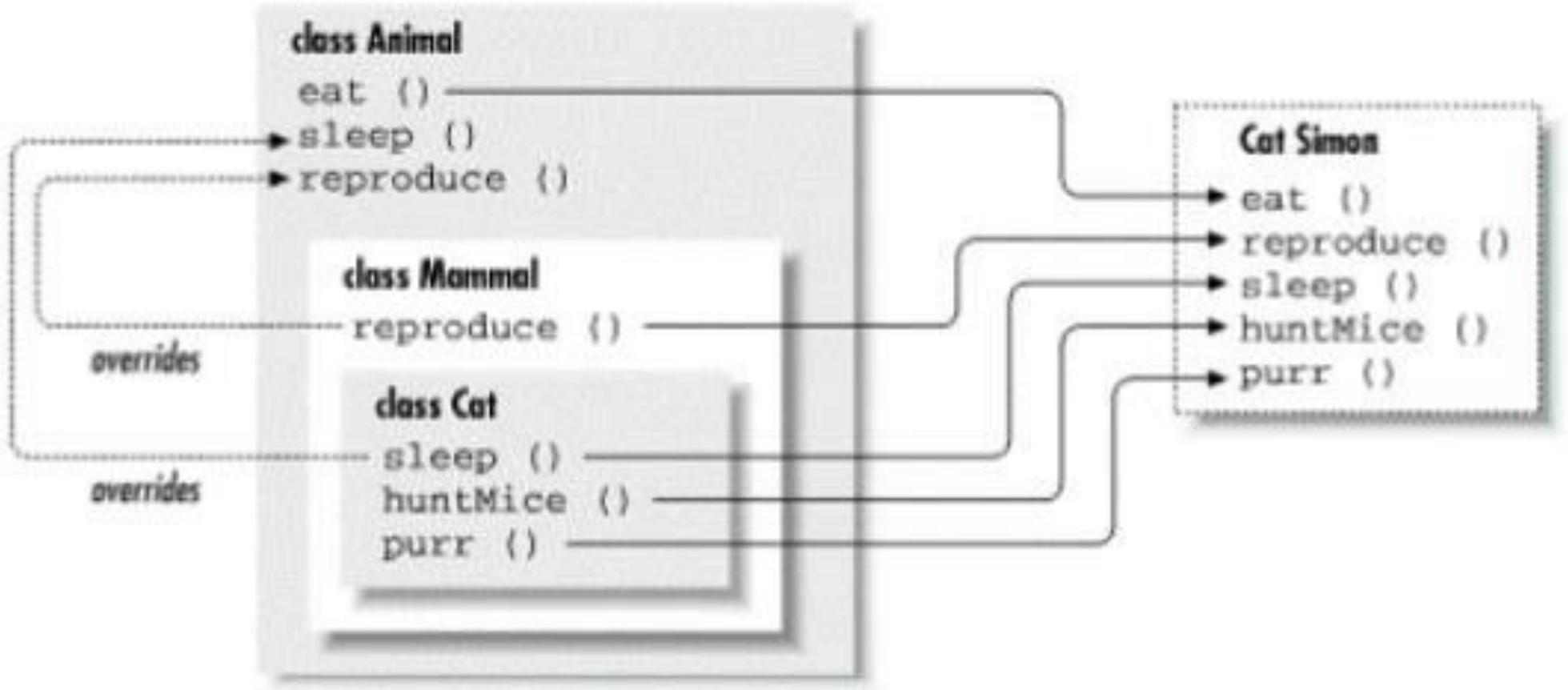
- **Класа**: не дозволува наследување на дадена класа

```
final class clsElCap extends clsFiksen{ . . . }
class clsBiPolarCap extends clsElCap{ . . . } //грешка!!!
```

- **Метод**: не дозволува да биде препокриен

```
class clsElCap extends clsFiksen { public final double getKapacitet() { return kapacitet; } }
class clsBiPolarCapExtends extends clsElCap{
  public double getKapacitet(){ . . .} //грешка!!!
 }
```

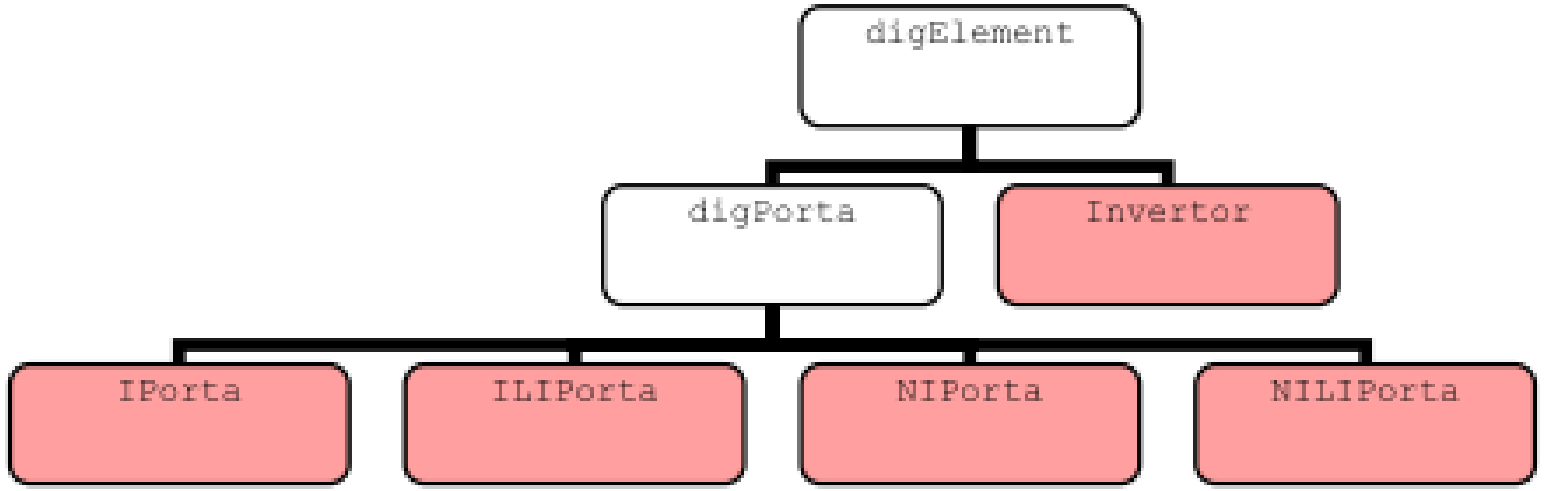
ПРЕПОКРИВАЊЕ НА МЕТОДИ



ПОЛИМОРФИЗАМ (ДИНАМИЧКО ПОВРЗУВАЊЕ)

- Полиморфизмот дозволува повторно искористување на кодот, но на поинаков начин од наследувањето

```
public static void main(){
  digElement[]  de= new digElement[3];
  de[0] = new IPorta();
  de[1] = new Invertor();
  de[2] = new ILIPorta();
}
```



- Објектните променливи имаат:
  - **Декларирaн тип** – наречен и статички тип
  - **Динамички тип** – типот на покажувачот за време на извршување
- **Методот кој се извршува над даден објект зависи само од неговиот динамички тип**

```
boolean izlez;
  for(int i=0;i<de.length();i++) izlez = de[i].reagiraj();
```

АПСТРАКТНИ КЛАСИ

- Апстрактните класи, за разлика од регуларните, не можат да **креираат** свои инстанци, но може да бидат наследени

```
abstract class logPorta{ . . . }

class IPorta extends logPorta{ . . . }

class ILIPorta extends logPorta{ . . . }

class NIPorta extends logPorta{ . . . }

class NILIPorta extends logPorta{ . . . }

logPorta lp;     //дозволено

lp = new IPorta() ;   //дозволено

lp = new logPorta();   /*класата е апстрактна и ова ќе  врати грешка */
```

АПСТРАКТНИ МЕТОДИ

```
class logPorta{
  abstract void reagiraj();   //апстрактен метод (нема тело)
        //самата класа станува апстрактна
 }
class IPorta extends logPorta  {
  void reagiraj(){ . . . }
// Методот мора да се препокрие. Во спротивно и оваа класа ќе стане апстрактна.
 }
```

КОМПОНЕНТИ НА КЛАСАТА

Component	Syntax	Description
Subclassing modifier (use only one)	**`abstract`**	Class must be extended to be useful.
	`final`	Class cannot be extended.
Access modifiers	**`public`**	Class is available outside of package.
	no access modifier	Class is available only within package.
Keyword **`class`**	**`class`** *`class-name`*	Class should be contained in a file called *`class-name.java`*.
`extends` clause	**`extends`** *`superclass-name`*	Indicates that this class is a subclass of the class *`superclass-name`* in the *`extends`* clause.
`implements` clause	**`implements`** *`interface-list`*	Indicates the interfaces that this class implements. The *`interface-list`* is a comma-separated list of interface names.
Class body	Enclosed in braces	Contains data fields and methods for the class.

ПОДАТОЧНИ ПОЛИЊА

Type of modifier	Keyword	Description
Access modifier (use only one)	`public`	Data field is available everywhere (when the class is also declared `public`).
	`private`	Data field is available only within the class.
	`protected`	Data field is available within the class, available in subclasses, and available to classes within the same package.
	No access modifier	Data field is available within the class and within the package.
Use modifiers (all can be used at once)	`static`	Indicates that only one such data field is available for all instances of this class. Without this modifier, each instance has its own copy of a data field.
	`final`	The value provided for the data field cannot be modified (a constant).

МЕТОДИ

Type of modifier	Keyword	Description
Access modifier (use only one)	`public`	Method is available everywhere (when the class is also declared as `public`).
	`private`	Method is available only within the class (cannot be declared `abstract`).
	`protected`	Method is available within the class, available in subclasses, and available to classes within the same package.
	No access modifier	Method is available within the class and to classes within the package.
Use modifiers (all can be used at once)	`static`	Indicates that only one such method is available for all instances of this class. Since a `static` method is shared by all instances, the method can refer only to data fields that are also declared `static` and shared by all instances.
	`final`	The method cannot be overridden in a subclass.
	`abstract`	The method must be overridden in a subclass.

ИНТЕРФЕЈСИ

- Софистициран начин за организација и контрола на објекти
- Шаблон (слично на header датотеки во C) - ги опишува задачите на одредена класа, без да специфицира како истите се извршуваат
- Една класа може да има повеќе интерфејси (слично со повеќекратно наследување во C++)

ИНТЕРФЕЈСИ

- Колекција на константи и апстрактни методи
- Не дефинира што прави методот
- Го дефинира само заглавието на методот (тип, име, параметри)
- Методите се секогаш **public** и **abstract**
- Променливите се **public**, **static** и **final**
- Се користи клучниот збор **interface** наместо **class**
- Се сместува во датотека **ime_na_interfejs.java**
- Мора да се имплементира во класа, за да може да се користи

ИНТЕРФЕЈСИ

```
public interface Complexity {
public void setComplexity (int complexity);
public int getComplexity();
}
```

- **Не** се креираат објекти од интерфејсот
- Тоа се прави во класите кои го имплементираат
- Една класа мора да ги имплементира сите методи од интерфејсот

```
Complexity x; // важи за референца
x = new Complexity(. . .); // Грешка
x = new Question (. . .); // Ова е во ред само ако Question го имплементира Complexity
```

ИНТЕРФЕЈСИ

```
public class Question implements Complexity {
private String question, answer;
private int complexityLevel;
public Question (String query, String result) { question = query;
answer = result;
complexityLevel = 1; }
public void setComplexity (int level) { complexityLevel = level; }
public int getComplexity() { return complexityLevel; }
public String getQuestion() { return question; }
public String getAnswer() { return answer; }
public boolean answerCorrect (String candidateAnswer) { return answer. equals(candidateAnswer); }
public String toString() { return question + "\n" + answer; }}
```

ИНТЕРФЕЈСИ

- Класата може да имплементира повеќе интерфејси

```
class ManyThings implements interface1, interface2, interface3 {
// сите методи од сите интерфејси
}
```

- Интерфејсите се пофлексибилни од апстрактните класи

  Апстрактната класа мора да е тесно поврзана со сите класи кои ќе наследуваат од неа

ГЕНЕРИЧКО ПРОГРАМИРАЊЕ

```
public int soberi (int [] niza)
   {
      int zbir=0;
      for(int i=0;i<niza.length; i++)
         zbir+=niza[i];
      return zbir;
   }
```

- Како програмата да работи со реални броеви?
  - Преоптоварување на фукции

ГЕНЕРИЧКО ПРОГРАМИРАЊЕ

- Програмирање независно од типот на податоци над кои се програмира (генерален (општ) код)
- Кодот е безбеден и лесен за читање
- Генеричко програмирање – програмирање со класи и методи кои имаат параметризирани типови на податоци
- Генерички класи
- Генерички методи

ГЕНЕРИЧКИ КЛАСИ

- Генеричките типови се моќна алатка за пишување на повтрно искористлив објектно-ориентиран код

```
class Pair <T> {
public T first;
public T second;
public Pair (T f, T s) { first = f; second = s; }
public Pair () { first = null; second = null; }
}
```

ГЕНЕРИЧКИ КЛАСИ

class Pair <T> { … }

- Инстанцирање на објект – замена на генеричкиот тип T
  - Pair <String> pair = new Pair <String> ("1","2");
- Резултатот сега е класа со конструктор:
  - public Pair (String f, String s) { … } ; ⇔ public Pair (T f, T s) { … }
- Pair <String> pair = new Pair <String> ("1",2); //грешка

ГЕНЕРИЧКИ КЛАСИ

- Може да постојат повеќе типови на параметри

```
class Pair <T, U> {
public T first;
public U second;
public Pair (T x, U y) { first = x; second = y; }
public Pair () { first = null; second = null; }
}
```

- Инстанцирање
  - Pair <String, Number> pair = new Pair <String, Number> ("1", 2);

ГЕНЕРИЧКИ МЕТОДИ

- Генерички методи може да се дефинираат и во рамки на обична класа

```
class Algorithms {
public static <T>  getMiddle (T [ ] a) {
return a[ a.length/2 ];
}. . .}
```

- Повикување на метод:
  - String s = Algorithms.<String>getMiddle (names);

ПРИМЕР

```
public class Holder<T> {
 private T value;
 public Holder(T a_value){
 value=a_value; }
 public T getValue(){
 return value; } }
Holder<String> h=new Holder<String>("String");
 Holder<Integer> hi=new Holder<Integer>(45);
 System.out.println(h.getValue());
 System.out.println(hi.getValue());
```

```
// izlez:
String
45
```