WORK ENVIRONMENT AND API

Work environments and code reuse

- The goal of software engineering is development **robust and reusable software - reusable code**
  can be **to realize** through **standardization** - **“plug and play” concept**
- Concept **working environment (** engl. **framework)** is based on the principles of "plug and play" and reusability
- Examples: Microsoft Office - Word, Excel, PowerPoint have most common menu items (File, Edit, View, Format ...), Windows - work with windows - work environment contains ready-made elements, **no need to constantly reinvent the wheel!**

- **The programmer uses the desktop to create applications by using ready-made interfaces**

Work environments and APIs

- **Advantages: a sense of consistency and uniformity, the programmer can use code that has already been written and tested**

- How are ready-made dialog boxes from the work environment used? There are rules set by the work environment, usually organized in documentation written by the creator of the environment, which leads to the notion **application programming interface (** engl.

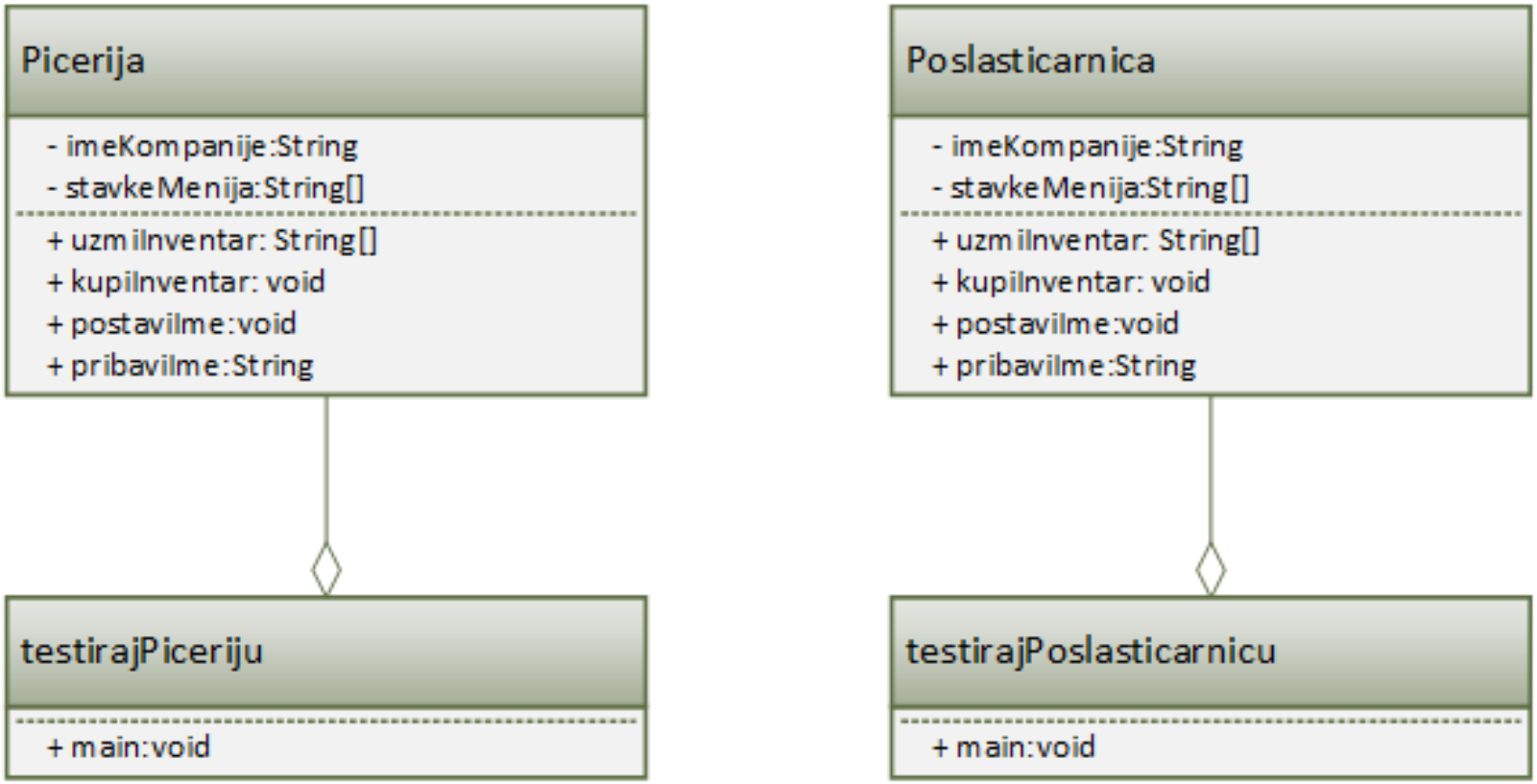
  Application Programming Interface - API)

- Using the API, you create "valid" applications using ready-made elements of the work environment and thus adapt to the standards - for example Java API, applet and web browsers

Example 3.5 - electronic business

- Development of a pizzeria website that enables online orders
- Our goal is to make **a working environment that would introduce the reusability of the code in practice**
- It needs to be developed **functional work environment** using **inheritance, merging, abstract classes** i **interface**
- What if we receive a request for the development of a pastry shop website the next day? Are we going to write a new application? How many more family shops can our web environment use?
- If we have **good and reliable working environment, programs** we could offer by **more favorable prices,** and yet they would be **already tested and applied,** what **reduces maintenance and troubleshooting work**

Example 3.5 - access without code reuse

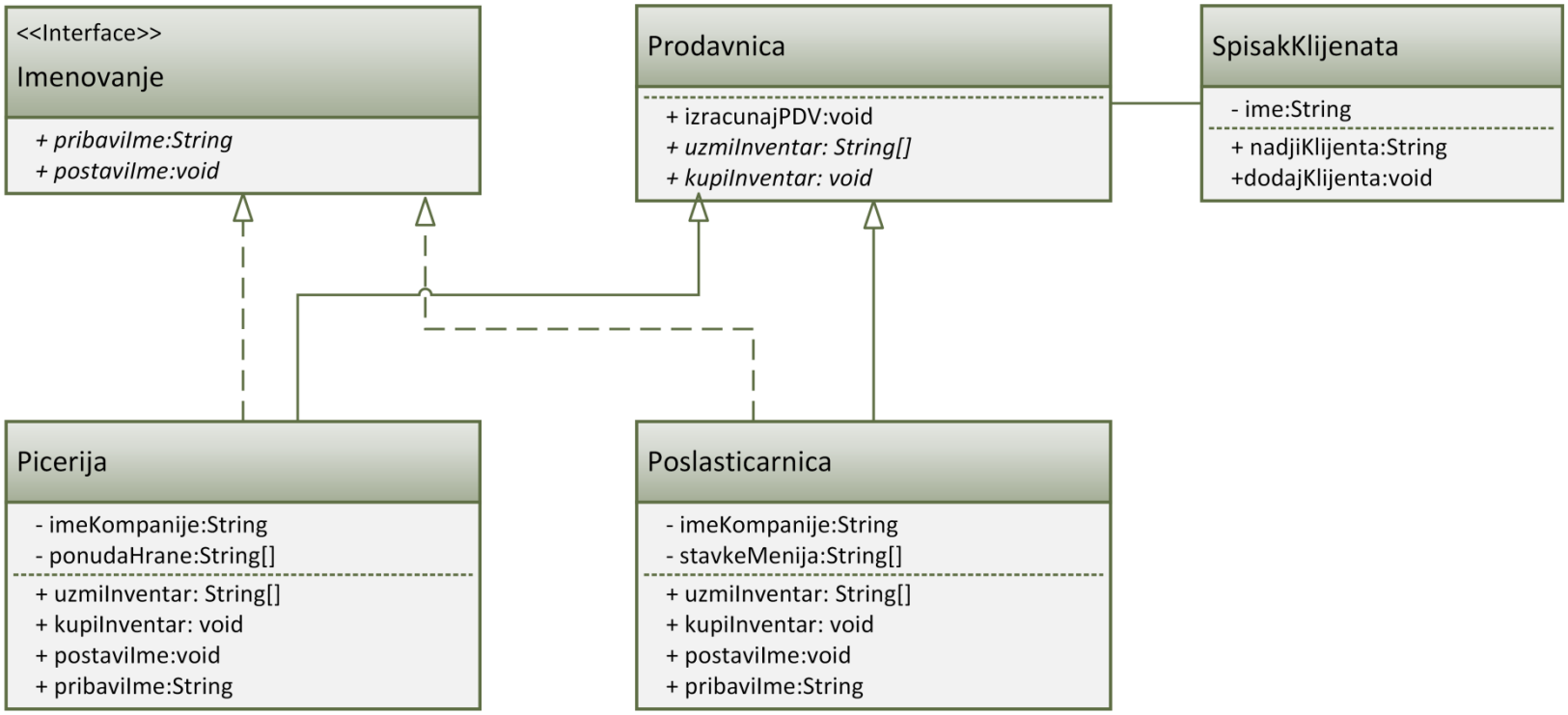


Example 3.5 - e-business solution

- We'll do it **abstract class** which **singles out a particular realization** i **an interface that models some behaviors**
- **The goal is a work environment in which we repeatedly use the code**
- Each specific application will be **bound only by contract,** there is no strict attachment to custom classes
- It consists of:
  - Interface Appointment, models behaviors, part of a contract,
  - Abstract classes Shop, singles out implementation, part of the contract,
  - Class Client List, which we use through merging and
  - New class realizations Shop for each client through the descendant classes that inherit it

Example 3.5 - class diagram

Abstract methods are labeled *italic* in words



Example 3.5 - abstract class Shop

```
public abstract class Shop {
        private Client List Client list ;

        public void calculate VAT () {
                System. out . println ( "The VAT rate is 20%!" );
        }

        public abstract String [] takeInventory ();

        public abstract void buyInventory (String article );
}
```

Example 3.5 - class Client List

```
public class Client List {
        private String [] name ;
        private int currentNumberClients ;
        private int maxNumberClients ;

        Client List () {}

        Client List ( int maxNumberClients ) {
                this . maxNumberClients = maxNumberClients ;
                this . currentNumberClients = 0;
                this . name = new String [ maxNumberClients ];
        }

        public String findClient (String name ) {
                for ( int i = 0; i < this . currentNumberClients ; i ++) {
                        if ( this . name [ i ] .equals ( name) ) {
                                return this . name [ i ];
                        }
                }
                return ( " The client was not found! " );
        }
        . . .
```

Example 3.5 - class Client List

```
. . .

public void addClient (String name ) {
        if ( this. currentNumberClients < this. maxNumberClients ) {
                this . name [ this . currentNumberClients ++] = name ;
        }
        else {
                System. out . println ( "There is no more space on the list
                                                clients! " );
        }
}
}
```

Example 3.5 - interface Appointment

```
public interface Appointment {

        String getName ();
        void setName (String name );

}
```

Example 3.5 - class Pizzeria

```
public class Pizzeria extends Shop implements Appointment {

        private String Company name ;

        private String [] food offer = {
                                "Pizza" ,,
                                "So what" ,,
                                "Salad" ,,
                                "Kalcona" ,,
                                "Juice" ,,
                                "Beer"
        };

        public String [] uzmiInventar () {
                return food offer ;
        }

        . . .
```

Example 3.5 - class Pizzeria

```
        . . .

        public void buyInventory (String article ) {System. out . println ( "\ nYou just ordered an item"

                                                                    +  article );
        }

        public String getName () {
                return Company name ;
        }

        public void setName (String name ) {
                Company name = name ;
        }

}
```

Example 3.5 - class Pastry shop

```
public class Pastry shop extends Shop implements Appointment {

        private String Company name ;

        private String [] Menu item = {
                                "Ice cream" ,,
                                "Cake" ,,
                                "Donut" ,,
                                "Coffee" ,,
                                "Tea" ,,
                                "Lemonade"
        };

        public String [] uzmiInventar () {
                        return Menu item ;
        }

        . . .
```

Example 3.5 - class Pastry shop

```
. . .

public void buyInventory (String article ) {System. out . println ( "\ nYou just ordered an item"

                                                    +   article );
}

public String getName () {
        return Company name ;
}

public void setName (String name ) {
        Company name = name ;
}

}
```

Example 3.5 - class Main

```
public class Main {
        public static void main (String [] args ) {

                Pastry shop Constantinople = new Confectionery (); Pizzeria hello = new Pizze
                ();

                Constantinople .setName ( "Europe" );
                hello .setName ( "Hello" );

                Constantinople .kupiInventar ( "Ice cream" );
                hello .kupiInventar ( "Pizza" );
                . . .
        }
}
```

Task for class work

- Task: to complete an abstract class Shop, interface Appointment, class Client List, as well as derived classes Pizzeria i Pastry shop new attributes and methods and expand the implementation of existing methods, so that the functionalities that could be required in everyday business are realized

- Realize new classes Restaurant i Bookstore which also inherit the abstract class Shop and implement the interface Appointment

- Test classes by creating multiple objects in the main program and calling the appropriate selected methods

ACCURACY AND ROBUSTNESS OF THE PROGRAM - GENERATION AND EXCEPTION PROCESSING