Prezentare laborator 6
Clase interne

Obiective

- Conceptul de clasă internă și motivația utilizării acestuia
- Prezentarea tipurilor de clase interne
  - Axare pe utilizarea clasele interne normale și anonime
- Folosire expresii lambda

Ce înseamnă o clasă internă

```
class Message {
   private MessageParserFromJson messageParserFromJson;
   private String text;

   public Message(File jsonFile) {
       messageParserFromJson = new MessageParserFromJson(jsonFile);
       text = messageParserFromJson.getMessageText();
   }

   private class MessageParserFromJson {
       private String text;

       MessageParserFromJson(File jsonFile) {
           //parse json file
       }

       String getMessageText() {
           return text;
       }
   }
}
```

De ce să folosim clase interne?

- Decizie de design
- Gruparea claselor și limitarea accesului la ele
- Separarea logicii unei clase și ascunderea detaliilor de implementare
- Explicație pe scurt pentru fiecare tip: link

Tipuri

- Clase interne normale (regular inner classes)
- Clase anonime (anonymous inner classes)
- Clase interne statice (static nested classes)
- Clase interne metodelor (method-local inner classes) sau blocurilor

Clase anonime

```
class Message {
    private MessageParserFromJson  messageParserFromJson;
    private String text;

    public Message(File jsonFile) {
        messageParserFromJson = new
MessageParserFromJson(jsonFile);
        text = messageParserFromJson.getMessageText();
    }

    private class MessageParserFromJson {
        private String text;

        MessageParserFromJson(File jsonFile) {
            //parse json file
        }

        String getMessageText() {
            return text;
        }
    }
}
```



```
interface MessageParserFromJson {
    String getMessageText();
}

class Message {
    private MessageParserFromJson  messageParserFromJson;
    private String text;

    public Message(File jsonFile) {
        messageParserFromJson = new MessageParserFromJson() {
            @Override
            public String getMessageText() {
                String parsedText =  "";
                if (jsonFile.exists()) {
                    // parse stuff
                }
                return parsedText;
            }
        };
        text = messageParserFromJson.getMessageText();
    }
}
```

Clase anonime - accesare variabile

```
interface MessageParserFromJson {
   String getMessageText();
}

class Message {
   private MessageParserFromJson  messageParserFromJson;
   private String text;

   public Message(File jsonFile) {
       StringBuilder defaultText =  new StringBuilder("");

       messageParserFromJson = new MessageParserFromJson() {
           @Override
            public String getMessageText() {
               String parsedText =  defaultText.toString();
               defaultText.append("smth");

                if (jsonFile.exists()) {
                    // parse stuff
                }
                return parsedText;
           }
       };
       text = messageParserFromJson.getMessageText();
   }
}
```

Java specs

"An anonymous class cannot access local variables in its enclosing scope that are not declared as final or effectively final."

In exemplul acesta nu folosim final si totusi compilează și rulează

Effectively final - introdus din Java 7

Clase anonime - accesare variabile

```
interface MessageParserFromJson {
   String getMessageText();
}

class Message {
   private MessageParserFromJson  messageParserFromJson;
   private String text;

   public Message(File jsonFile) {
       StringBuilder defaultText =  new StringBuilder("");

       messageParserFromJson = new MessageParserFromJson() {
           @Override
            public String getMessageText() {
               String parsedText =  defaultText.toString();
               defaultText.append("smth");

                if (jsonFile.exists()) {
                    // parse stuff
                }
                return parsedText;
           }
       };
       defaultText =  new StringBuilder(""); // nu mai este effectively final si codul nu mai compileaza
       text = messageParserFromJson.getMessageText();
   }
}
```

Diverse detalii

- Clasele interne sunt membri ai claselor ca și metodele și câmpurile
  - au modificatorii de acces: **private, protected, default, public**
  - pot avea modificatorul **static**
- Pot accesa membrii privați ai claselor
- Fișiere .class generate pentru fiecare clasă internă, inclusiv cele anonime
- Obiectul clasei exterioare
  - instanțiere
  - moștenire
- Pentru nested classes (interne statice) nu este nevoie de obiect al clasei exterioare

Expresii lambda

- Concept din programarea funcțională
- Metode fără nume
- Sintaxă

```
(parameter1, parameter2,...) -> { code block }
```

- Clasele anonime care suprascriu singură metodă pot fi transformate în lambda

Expresii lambda - exemplu

```
ArrayList<Integer> values =new ArrayList<>(Arrays.asList(1, 2, 3, 4, 5));

values.removeIf((v) -> v %2 == 0);

values.forEach((v) -> System.out.println(v));

values.forEach(System.out::println);
```

Expresii lambda - clase anonime

```
interface MessageParserFromJson {
   String getMessageText(File jsonFile);
}
class Message {

   private MessageParserFromJson  messageParserFromJson;
   private String text;

   public Message(File jsonFile) {
       messageParserFromJson = new MessageParserFromJson() {
          @Override
          public String getMessageText(File jsonFile) {
              String parsedText =  "";
              // parse stuff
              return parsedText;
           }
       };


       text = messageParserFromJson.getMessageText(jsonFile);


   }

}
```

```
interface MessageParserFromJson {
   String getMessageText(File jsonFile);
}
class Message {

   private MessageParserFromJson  messageParserFromJson;
   private String text;

   public Message(File jsonFile) {
       messageParserFromJson = jsonFile1 -> {
           String parsedText =  "";
           // parse stuff from jsonFile1
           return parsedText;
       };

       text = messageParserFromJson.getMessageText(jsonFile);
   }

}
```