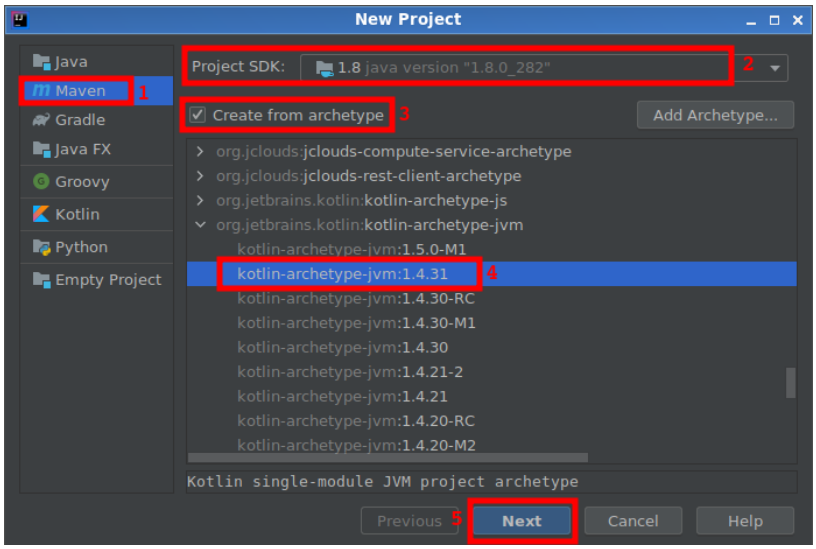
Paradigme de Programare - Laborator 3

Aplicații simple ale ADT în Kotlin

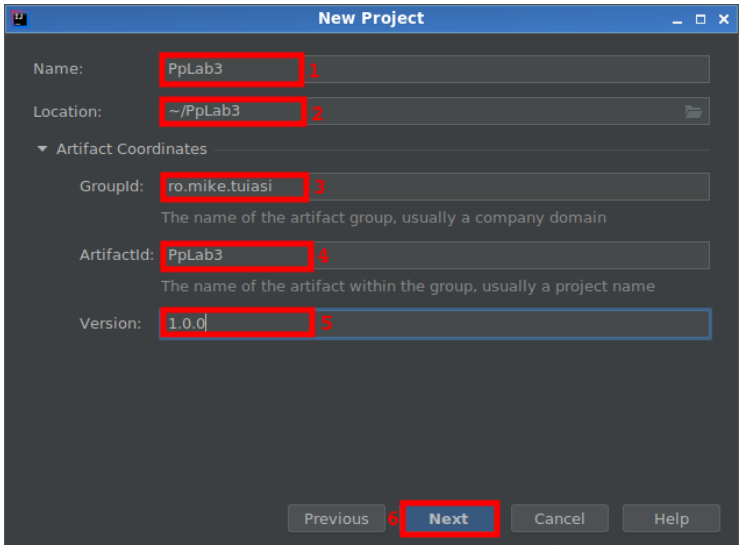
Introducere

Crearea unui proiect Maven

File → New → Project → Maven → se bifează „Create from archetype” și se selectează **org.jetbrains.kotlin:kotlin-archetype-jvm → kotlin-archetype-jvm:1.4.31** → Next.



Apoi se completează denumirea proiectului, se alege locația unde se dorește a fi salvat și se setează (opțional) id-urile pentru grup, artefact, precum și versiunea proiectului.



La final se apasă Next, iar pe fereastra următoare, Finish.

Adăugarea dependențelor Maven

Proiectele Maven au un fișier numit *pom.xml* (Project Object Model) care conține informații despre proiect precum și configurările utilizate de Maven pentru a da build proiectului (tipul de împachetare: jar/war, dependențele, plugin-urile, repozitoriile, configurări de compilare, etc).

Se recomandă parcurgerea următoarelor resurse:

- https://www.w3schools.com/cssref/css_selectors.asp

- https://khttp.readthedocs.io/en/latest/
- https://jsoup.org/cookbook/

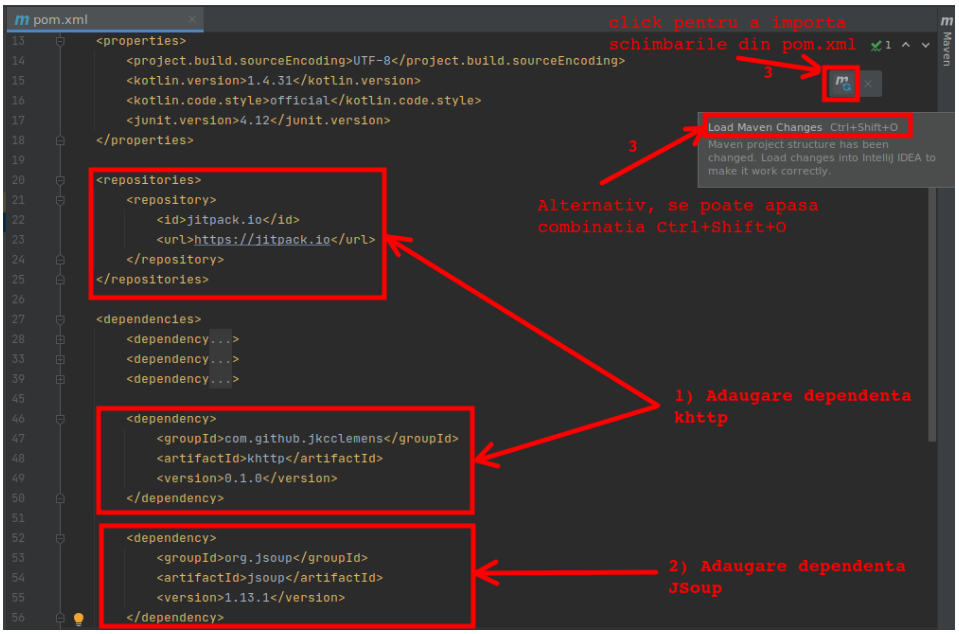
În cadrul acestui laborator se vor adăuga două dependențe:

- **khttp** - pentru cereri de tip HTTP (get, post, put, delete)
- **JSoup** - pentru procesarea unor fișiere de tip XML, cu precădere HTML-uri.

Se deschide fișierul pom.xml din proiectul creat și se adaugă în interiorul tag-ului <project>, următoarele linii:

```
<repositories>
    <repository>
        <id>jitpack.io</id>
        <url>https://jitpack.io</url>
    </repository>
</repositories>

<dependencies>
    <!-- ... -->
    <dependency>
        <groupId>com.github.jkcclemens</groupId>
        <artifactId>khttp</artifactId>
        <version>0.1.0</version>
    </dependency>
    <dependency>
        <groupId>org.jsoup</groupId>
        <artifactId>jsoup</artifactId>
        <version>1.13.1</version>
    </dependency>
    <!-- ... -->
</dependencies>
```



Expresii regulate

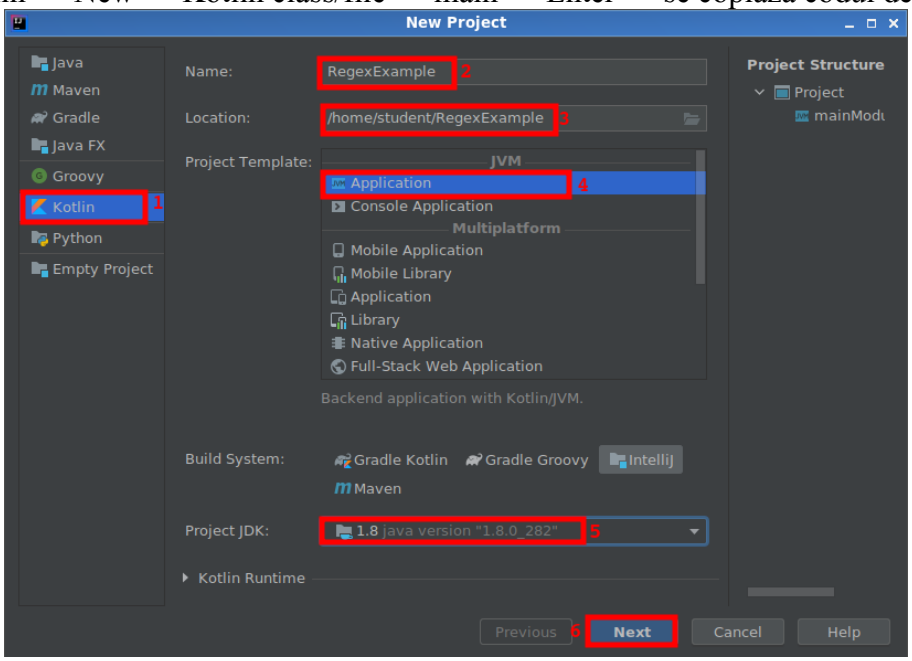
O expresie regulată este o secvență de caractere care definește un tipar de căutare. Acestea sunt utile pentru operații de căutare în principal, dar și pentru înlocuire sau pentru împărțirea unui string într-o listă de substring-uri în funcție de tiparul ales.

Pentru mai multe detalii, vezi următoarele resurse:

- https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.text/-regex/
- cartea „Regex Quick Syntax Reference: Understanding and using Regular Expressions" scrisă de Zsolt Nagy
- cartea „Regular Expressions Cookbook" (ediția a doua) scrisă de Jan Goyvaerts și Steven Levithan
- https://regex101.com/
- cheatsheet-ul anexat laboratorului

Exemplul 1: utilizarea expresiilor regulate

Pentru acesta, se poate crea un proiect Kotlin simplu: File → New → Project → Kotlin → se selectează JDK-ul Java 1.8 → Next → Finish. Se dă click dreapta pe folder-ul src/main/kotlin → New → Kotlin class/file → main → Enter → se copiaza codul de mai jos.




```
fun printAllRegexMatches(regex: Regex, searchString: String) {
    for(item in regex.findAll(searchString)) {
        println(item.value)
    }
    println("-".repeat(100))
}

fun main() {
    val testString = "link/ether a0:b1:c2:d3:e4:f5 brd ff:ff:ff:ff:ff:ff\n" +
                     "inet 192.168.0.2/24 brd 192.168.0.255 scope global eno1\n" +
                     "Hi,\n You can contact me at john.smith@gmail.com\n" +
                     "You should use a search engine like www.duckduckgo.com\n" +
                     "I'll meet you at 08:00 AM tomorrow"
    val stringWithDuplicates = "one two two three three three four four four four"
```

```
    val ipRegex = Regex("((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.){3}(25[0-
5]|2[0-4][0-9]|[01]?[0-9][0-9]?)")
    val emailRegex = Regex("\\w+[+.\\w-]*@([\\w-]+.)*\\w+[\\w-]*.([a-
z]{2,4}|\\d+)")
    val urlRegex = Regex("(https?:\\/\\/)?www\\.[-a-zA-Z0-
9@:%._\\+~#=]{2,256}\\.[a-z]{2,6}\\b([-a-zA-Z0-9@:%_\\+.~#?&//=]*)")
    val timeRegex = Regex("(0?[1-9]|1[0-2]):[0-5][0-9]")
    val duplicatesRegex = Regex("(\\b\\w+\\b)(?=.*\\b\\1\\b)")

    // Find all matches
    for(regex in listOf(ipRegex, emailRegex, urlRegex, timeRegex)) {
        printAllRegexMatches(regex, testString)
    }

    // Match string:
    println("'08:00' matchEntire:
${timeRegex.matchEntire("08:00")?.value.orEmpty()}")
    println("'Tomorrow at 09:15' matchEntire:
${timeRegex.matchEntire("Tomorrow at 09:15")?.value.orEmpty()}")
    println("'Tomorrow at 09:16' matches: ${timeRegex.matches("Tomorrow at
09:16")}")
    println("'Tomorrow at 09:17' containsMatchIn:
${timeRegex.containsMatchIn("Tomorrow at 09:17")}")

    // Remove duplicates:
    println("Without duplicates:
${duplicatesRegex.replace(stringWithDuplicates, "")}")

    // Find first match:
    println("First duplicate:
${duplicatesRegex.find(stringWithDuplicates)?.value.orEmpty()}")

    // Split by regex:
    println("Regex split:
${Regex("\\d+").split("This10text20is30splitted40by50regex")}")
}
```

Exemplul 2: cereri HTTP de tip GET și parsarea unui HTML

După crearea proiectului conform indicațiilor din introducere, se expandează folder-ul *src/main/kotlin/ro.mike.tuiasi* → se dă click dreapta pe acesta → New → Kotlin Class/File → HTMLParser → Enter. Se copiază codul de mai jos:

```
package ro.mike.tuiasi

import org.jsoup.Jsoup
import org.jsoup.nodes.Document
import org.jsoup.select.Elements
import java.io.File

/**
 * @param url - Uniform Resource Locator - address of an website
 * @return HTML content corresponding to the URL as a string
 */
```

```
fun testKhttpGetRequest(url: String) : String {
    val response = khttp.get(url)
    println("${response.statusCode}\t ${response.headers["Content-Type"]}")
    return response.text
}

/**
 * @param source - string specifying the source type (url, file, string)
 * @param url - string containing an URL, a path to a HTML file or an HTML string
 * @param baseURI - string used for the relative links inside of a local HTML file
 * @throws Exception - if the source is unknown
 */
fun testJsoup(source: String, url: String, baseURI: String?=null) {
    var htmlDocument: Document? = null
    htmlDocument = when(source) {
        "url" -> Jsoup.connect(url).get()
        "file" -> Jsoup.parse(File(url), "UTF-8", baseURI)
        "string" -> Jsoup.parse(url)
        else -> throw Exception("Unknown source")
    }

    val cssHeadlineSelector: String = "#khttp-http-without-the-bullshit h1"
    val cssParagraphSelector = "#khttp-http-without-the-bullshit p"
    val cssLinkSelector = "#khttp-http-without-the-bullshit > p > a"
    println(htmlDocument.title())
    println(htmlDocument.select(cssHeadlineSelector).text())
    val paragraphs: Elements = htmlDocument.select(cssParagraphSelector)
    for (paragraph in paragraphs) {
        println("\t${paragraph.text()}")
    }
    val links = htmlDocument.select(cssLinkSelector)
    println("-".repeat(100))
    for (link in links) {
        println("${link.text()}\n\t${link.absUrl("href")}")
    }
}

fun main() {
    val projectPath: String = System.getProperty("user.dir")
    val htmlPath: String = "${projectPath}/src/main/resources/example.html"
    val url: String = "https://khttp.readthedocs.io/en/latest/"
    val htmlContent: String = testKhttpGetRequest(url)
    println("*".repeat(100))
    testJsoup("url", url)
    println("*".repeat(100))
    testJsoup("file", htmlPath, "mike.tuiasi.ro")
    println("*".repeat(100))
    testJsoup("string", htmlContent)
}
```

Se creează de asemenea un nou pachet: click dreapta pe folder-ul *src/main* → New → Directory → resources → Enter. Apoi, click dreapta pe pachetul *resources* → New → HTML file → HTML 5 file → example.html → Enter. Se înlocuiește conținutul cu liniile de mai jos:

```
<!DOCTYPE html>
```

```
<html lang="ro">
<head>
    <meta charset="UTF-8">
    <title>Laboratorul 3 - Paradigme de programare</title>
</head>
<body>
<h4 id="first">Acesta este un exemplu simplu de fisier HTML care trebuie parsat</h4>
<div>
    <p class="custom">Acest paragraf trebuie extras utilizand un CSS selector pentru
clasa custom</p>
</div>
</body>
</html>
```

Exemplul 3: agendă de telefon cu operații CRUD

Să se implementeze o agendă simplă de telefon care să pună la dispoziție operațiuni de adăugare, căutare și ștergere.

```
class Birth(val year: Int, val Month: Int, val Day: Int){
    override fun toString() : String{
        return "($Day.$Month.$year)"
    }
}

class Contact(val Name: String, val Phone: String, val BirthDate: Birth){
    fun Print() {
        println("Name: $Name, Mobile: $Phone, Date: $BirthDate")
    }
}

fun main(args : Array<String>){
    val agenda = mutableListOf<Contact>()
    agenda.add(Contact("Mihai", "0744321987", Birth(1900, 11, 25)))
    agenda += Contact("George", "0761332100", Birth(2002, 3, 14))
    agenda += Contact("Liviu" , "0231450211", Birth(1999, 7, 30))
    agenda += Contact("Popescu", "0211342787", Birth(1955, 5, 12))
    for (persoana in agenda){
        persoana.Print()
    }
    println("Agenda dupa eliminare contact [George]:")
    agenda.removeAt(1)
    for (persoana in agenda){
        persoana.Print()
    }
    agenda.remove(Contact("Liviu" , "0231450211", Birth(1999, 7, 30)))
    println("Agenda dupa eliminare contact [Liviu]:")
    agenda.removeAt(1)
    for (persoana in agenda){
        persoana.Print()
    }
}
```

Exemplul 4: translator trivial bazat pe colecții de tip Map și List

Să se implementeze un translator trivial bazat pe un dicționar de coduri. Pentru aceasta se vor utiliza colecții de tip Map și List.

```
fun main(args : Array<String>){
    val Dictionar = hashMapOf<String, String>(
        "Once"                to  "Odata",
        "upon"                to  "ca",
        "a"                   to  "",
        "time"                to  "niciodata",
        "there"               to  "acolo",
        "was"                 to  "a fost",
        "an"                  to  "o",
        "old"                 to  "batrana",
        "woman"               to  "femeie",
        "who"                 to  "care",
        "loved"               to  "iubea",
        "baking"              to  "sa gateasca",
        "gingerbread"         to  "turta dulce",
        "She"                 to  "Ea",
        "would"               to  "ar fi",
        "bake"                to  "gatit",
        "gingerbread"         to  "turta dulce",
        "cookies"             to  "biscuiti",
        "cakes"               to  "prajituri",
        "houses"              to  "case",
        "and"                 to  "si",
        "people"              to  "oameni",
        "all"                 to  "toti",
        "decorated"           to  "decorati",
        "with"                to  "cu",
        "chocolate"           to  "ciocolata",
        "peppermint"          to  "menta",
        "caramel"             to  "caramel",
        "candies"             to  "bomboane",
        "colored"             to  "colorate",
        "ingredients"         to  "ingrediente"
    )

    val Poveste = "Once upon a time there was an old woman who loved baking
gingerbread. She would bake gingerbread cookies, cakes, houses and
gingerbread people, all decorated with chocolate and peppermint, caramel
candies and colored ingredients."

    val words1 = Poveste.split(" ")

    println("Cuvintele din poveste [${words1.count()}]:")
    for (word in words1)
        print(word + " ")

    val words2 = mutableListOf<String>()
    for (word in words1){
        words2.add(word.trim(',','.'))
    }
```

```
    println("\n")
    println("Povestea tradusa ar suna cam asa:")
    for (item in words2){
        if (Dictionar.contains(item))
            print(Dictionar[item])
        else
            print("[$item]")
        print(" ")
    }
}
```

Exemplul 5: Histograma distribuției cuvintelor într-un text

Se va utiliza abilitatea compilatorului Graal de a mixa limbaje. În acest caz se cere histograma distribuției cuvintelor dintr-un text. Dacă implementarea s-ar fi realizat utilizând numai Kotlin efortul de programare ar fi fost dublu, acesta fiind eliminat prin utilizarea funcționalităților din limbajul R.

HINTS: Se pornește de la un proiect Kotlin JVM folosind GraalVM ca JDK, la care au fost adăugate două fișiere:

1. Histogram.kt
2. RHistogram.java

Din java se vor utiliza următoarele biblioteci:

```
import static java.awt.image.BufferedImage.TYPE_INT_RGB;
import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import org.graalvm.polyglot.Context;
import org.graalvm.polyglot.Value;
```

Nu uitați să denumiți fișierul după clasa principală (sau să verificați dacă regula este respectată). Se vor adăuga și două variabile pentru gestiunea dimensiunii suprafeței de desenare (plot)

```
public class RHistogram {
    //dimensiunea suprafetei in care plotam
    private static final int WIDTH = 1000;
    private static final int HEIGHT = 500;
```

Apoi se va adăuga o metodă pe care o vom folosi pentru a scrie imagini PNG desenate din R. Parametrul E[] values este un array generic care ne permite să menținem o singură implementare a acestei metode, în timp ce vom trimite array-uri cu tip diferit, din Kotlin.

```
public static <E> void WriteImage(Value showPlot, String fname, E[] values){
    //construim o imagine in care vom plota graficul
    BufferedImage image = new BufferedImage(WIDTH, HEIGHT, TYPE_INT_RGB);
    Graphics2D graphics = (Graphics2D) image.getGraphics();
    graphics.setBackground(new Color( r: 255,  g: 255,  b: 255));
    graphics.clearRect( x: 0,  y: 0, WIDTH, HEIGHT);
    //executam functia R, evaluata in showPlot, cu parametrii corecti pentru grafic
    //si array-ul values primit ca parametru in aceasta metoda Java, din Kotlin
    showPlot.execute(graphics, WIDTH, HEIGHT, values);

    //construim fisierul PNG folosind numele dat ca parametru
    try {
        ImageIO.write(image,  formatName: "png", new File( pathname: fname+ ".png"));
    } catch (IOException e) {
        e.printStackTrace();
    }
    System.out.println("SUCCESS : [" + fname + "]");
}
```

Se va adăuga și metoda *BuildHistogram* pe care o vom apela ulterior din Kotlin pentru a construi două tipuri de grafice:

1. Histograme de cuvinte considerând **values** ca un String[]: how = true
2. Grafice de puncte XY, considerând **values** ca Int[] : how = false.

```
//functia care construieste graficul, primeste 2 parametri, un array de cuvinte si un nume de
//fisier in care scriem imaginea
public static<S> void BuildHistogram(S[] values, String fname, boolean how) {
    //construim un context Polyglot pentru R
    Context context = Context.newBuilder( ...permittedLanguages: "R").allowAllAccess(true).build();
    String src;
    if (how) {
        src =
                "library(lattice); " +                  //GraalVM stie sa foloseasca momentan libraria lattice si grid pentru plotare
                        "function(g, w, h, data) { " +  //functia R primeste 4 parametri : suprafata grafica din java, Width, Height si array-ul de cuvinte
                        "   grDevices:::awt(w,h,g);" +  //folosim Java awt pentru afisare
                        "   B <- c(data);" +            //construim un array R dintr-un array Java
                        "   tab <- table(B);" +         //convertim la table, array-ul B. Operatia va produce o histograma cu perechi val_unica -> frecventa
                        "   print(barchart(tab));" +    //afisam histograma folosind functia barchart(), mai corect ar fi fost barplot() insa nu functioneaza momentan
                        "   dev.off();" +               //inchidem suprafata
                        "}";
    }else{
        src =
                "library(lattice); " +                  //GraalVM stie sa foloseasca momentan libraria lattice si grid pentru plotare
                        "function(g, w, h, data) { " +  //functia R primeste 4 parametri : suprafata grafica din java, Width, Height si array-ul de cuvinte
                        "   grDevices:::awt(w,h,g);" +  //folosim Java awt pentru afisare
                        "   B <- c(data);" +            //construim un array R dintr-un array Java
                        "   x <- 0:(length(B) - 1);" +  //construim elementele pe axa X
                        "   print(xyplot(B ~ x));" +    //afisam histograma folosind functia barchart(), mai corect ar fi fost barplot() insa nu functioneaza momentan
                        "   dev.off();" +               //inchidem suprafata
                        "}";
    }
    //evaluam functia R
    Value showPlot = context.eval( languageId: "R", src);

    WriteImage(showPlot, fname, values);
}
```

Metoda se termină cu un apel la funcția care scrie imaginea PNG. Atenție, deși limbajul R suportă mai multe funcții pentru desenarea graficelor pentru moment GraalVM exista suport doar pentru bibliotecile **lattice** și **grid**.

După aceea vom adăuga și funcția *main()* în fișierul Kotlin. Vom folosi *java.io.File* pentru a deschide și citi din fișierul cu numele "Fisier.txt". Se vor construi simultan două liste:

1. **trim_words** care este o listă de cuvinte (numai cu litere mici), din care s-au eliminat restul separatorilor de text (e.g. semnele de punctuație periferice)
2. **chars** care reprezintă o listă de litere mari, ce este creată respectând ordinea în care acestea apar în cuvintele din **trim_words**.

La sfârșitul funcției main() apelăm metoda *BuildHistogram* din clasa Java *RHistogram*, pentru a crea cele două fișiere PNG care conțin histogramele cuvintelor și a caracterelor unice.

```
import java.io.File

//functia main()
fun main(args : Array<String>){
    //citim liniile din fisier
    val lines = File( pathname: "Fisier.txt").reader().readText()
    //construim un array de cuvinte, separand prin spatiu
    val words = lines.split( ...delimiters: " ")

    //eliminam semnele de punctuatie de pe marginile cuvintelor
    val trim_words = mutableListOf<String>()
    words.forEach { it: String
        val filter = it.trim( ...chars: ',','.','"','?', '!')
        trim_words += filter.toLowerCase()
        print(filter + " ")
    }
    println("\n")

    //construim o lista cu toate caracterele folosite 'A..Z'
    val chars = mutableListOf<String>()
    trim_words.forEach { it: String
        for (c in it){
            if (c in 'a'..'z' || c in 'A'..'Z') {
                chars += c.toUpperCase().toString()
                print(c.toUpperCase())
            }
        }
    }
    println("\n")

    //construim histograma pentru cuvinte
    RHistogram.BuildHistogram(trim_words.toTypedArray(), fname: "Words", how: true)
    //construim histograma pentru caractere
    RHistogram.BuildHistogram(chars.toTypedArray(), fname: "Chars", how: true)
}
```

Pentru construcția histogramelor, R folosește un mecanism prin care se asociază caracterelor unice, numărul total de apariții (frecvența).

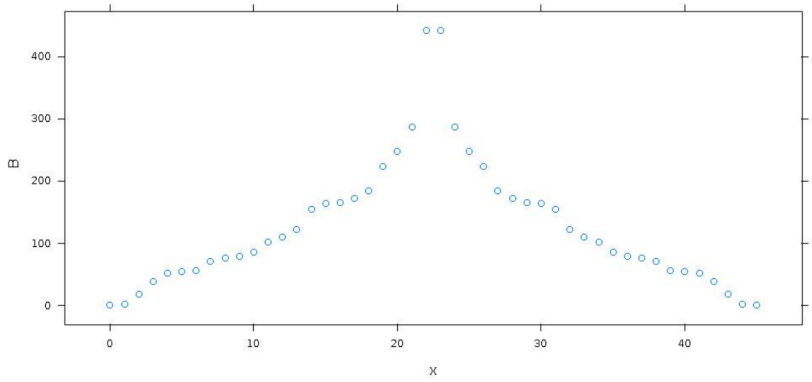
Template-ul celor 3 funcții pe care trebuie să le completați este prezentat mai jos.

```
fun GetUniqueWordCount(all_words : List<String>) : MutableMap<String, Int> {
    //functia pentru calculul cuvintelor unice
    val result : MutableMap<String, Int> = mutableMapOf<String, Int>()
    return result
}

fun GetUniqueCharCount(all_chars : List<String>) : MutableMap<Char, Int> {
    //functia pentru calculul caracterelor unice
    val result : MutableMap<Char, Int> = mutableMapOf<Char, Int>()
    return result
}

fun SortByHitCount(items : MutableMap<Char, Int>, how: Boolean) : MutableMap<Int, Char>{
    //functia de sortare a caracterelor, dupa valoare (frecventa), atat crescator cat si descrescator, in functie de how
    val result : MutableMap<Int, Char> = mutableMapOf<Int, Char>()
    return result
}
```

Pentru punctul trei ar trebui să adăugați niște cod suplimentar în *main()* și să aveți un al treilea apel la *RHistogram.BuildHistogram*(…) astfel încât să obțineți următorul grafic:



Aplicaţii şi teme

Aplicaţii de laborator:

1. Pornind de la exemplul 3:
   - să se implementeze operația de căutare (după nume/număr de telefon);
   - să se implementeze operația de actualizare a numărului de telefon;
   - să se extragă diagrama asociată de clasă incluzând și facilitățile cerute suplimentar ca temă de laborator.
2. Pornind de la exemplul 4 se cere adăugarea de noi facilități:
   - Adăugarea de cuvinte la dicționar;
   - Salvarea în fișier a poveștii traduse;
   - Inițializarea dicționarului prin citirea lui dintr-un fișier;
   - Extragerea unui dicționar dintr-un fișier text de tip ebook.

   Observație: studenții pot propune/implementa alte facilități, atât timp cât sunt relevante în contextul acestei probleme.
3. Pornind de la exemplul 5:
   - Construiți în Kotlin același mecanism de măsurare a frecvenței elementelor unice și afișați cuvintele unice din *trim_words;*
   - Construiți în Kotlin același mecanism de măsurare a frecvenței elementelor unice și afișați cuvintele unice din chars;
   - Pentru frecvențele caracterelor unice calculate anterior să se implementeze următoarele:
     - Afișarea perechilor (frecvență → caracter) sortate crescător și descrescător;
     - Afișarea graficelor variației de frecvență sortate anterior crescător și descrescător și concatenarea lor într-un grafic de puncte.

Teme pe acasă:

1. Utilizând biblioteca JSoup, să se proceseze un RSS feed (de exemplu: http://rss.cnn.com/rss/edition.rss) construind un ADT pentru stocarea datelor. ADT-ul trebuie să conțină o listă de elemente (item-uri; un item poate fi un ADT separat), precum și atributele generale (title, link, description, pubDate). La final, se va afișa titlul și link-ul fiecărui item.

   Mai jos aveți o structură generică a unui RSS feed:

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0">
  <channel>
    <title>PP - Laborator 3</title>
    <link>http://mike.tuiasi.ro/LabPP3.pdf</link>
    <description>Aplicatii simple ale ADT in Kotlin</description>
    <item>
      <title>Crearea unui proiect Maven</title>
      <link>http://mike.tuiasi.ro/LabPP3.pdf</link>
      <description>Adaugarea dependentelor in pom.xml</description>
    </item>
    <item>
      <title>Expresii regulate</title>
      <link>http://mike.tuiasi.ro/LabPP3.pdf</link>
      <description>Exemple de expresii regulate in Kotlin</description>
    </item>
  </channel>
</rss>
```

2. Utilizând colecțiile și genericele să se realizeze un program consolă pentru procesarea ebook-urilor în format text care va efectua următoarele:

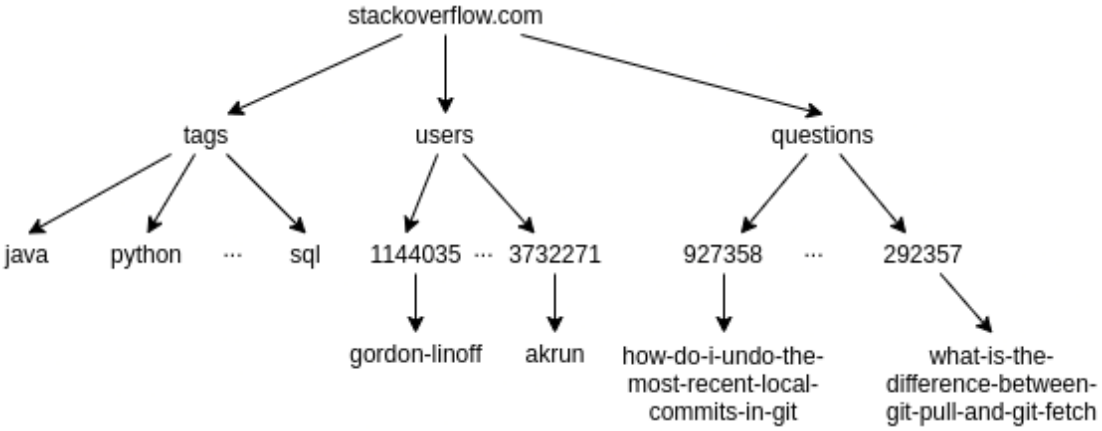
- **Must have:**
  - Eliminarea spațiilor multiple din text (rămâne numai unul).
  - Eliminarea salturilor la linie nouă multiple din text.
  - Detectarea și eliminarea numărului de pagină (hint: este cuprins între multe spații albe)
- **Nice to have:**
  - Detectarea și eliminarea numelui autorului (hint o secvență de două sau mai multe cuvinte care se repetă în cadrul textului și este separată de restul textului de mai multe spații albe și/sau caractere de control).
  - Detectarea și eliminarea numelui capitolului.
  - Identificarea setului de caractere românești și schimbarea lui cu cel utilizat actual în caz că este maparea veche.

Hint: se pot folosi expresiile regulate.

Observație: Studentul poate alege să implementeze tema utilizând numai limbajul Kotlin sau poate să își ușureze munca folosind orice mix de limbaje dintre cele suportate de GraalVM (e.g. python, kotlin, R).

3. Să se implementeze o aplicație în Kotlin care să extragă toate link-urile de la un URL dat cu o **recursivitate de ordin 2** (se iau întâi toate link-urile de la URL-ul inițial, apoi pentru fiecare link extras, se extrag din nou link-urile, păstrându-le numai pe cele care indică spre același domeniu). Pentru aceasta se va utiliza biblioteca **Jsoup** și eventual **khttp**. După extragerea tuturor acestor link-uri, se va crea un **arbore** pornind de la denumirea domeniului (vezi figura de mai jos). După alcătuirea arborelui, se vor crea două funcții:
   - serializeTree - care va serializa arborele (arborele va fi scris sub formă de string sau într-un fișier text)
   - deserializeTree - care va deserializa arborele (arborele se va reface dintr-un string sau dintr-un fișier text)

   Modalitatea de serializare/deserializare este la alegerea studentului. Nu este obligatoriu să fie generică (valabilă pentru orice URL). Totuși, modalitățile de serializare generice și/sau cele optime vor primi puncte bonus.



O variantă posibilă de serializare a arborelui într-un fișier:

```
stackoverflow.com: tags, users, questions
stackoverflow.com/tags: java, python, ..., sql
stackoverflow.com/users: 1144034, ..., 3732271
stackoverflow.com/users/1144034: gordon-linoff
stackoverflow.com/users/3732271: akrun
stackoverflow.com/questions: 927358, ..., 292357
stackoverflow.com/questions/927358: how-do-i-undo-the-most-recent-local-commits-in-git
stackoverflow.com/questions/292357: what-is-the-difference-between-git-pull-and-git-fetch
```

Anexe

1. În cazuri “disperate” (lapsus/lipsă de buffer la curs) cu privire la unele din apelurile din codurile exemplu utilizate în lucrarea de laborator, studentul se poate duce la https://kotlinlang.org/docs/reference/ unde prin utilizarea funcției de căutare disponibile are acces la documentația suport pusă la dispoziție de creatorii limbajului.

2. În tabelul 1 găsiți o comparație rapidă privind echivalențele specifice paradigmei nestructurate între CPP, Java și Kotlin.

C/C++	Java	Kotlin
int x;	int x;	val x : Int var x : Int
string x[2]={“str1”, “str2”};	String[] x = {“str1”, “str2”};	val x : Array<String>
double x = (4.5 > 1.2)? 7.3:9.1;	Double x = (4.5 > 1.2)? 7.3:9.1;	val x = if (4.5 > 1.2) 7.3 else 9.1
int x=6; if (x>5) x ^= 4;	int x = 6; if (x>5) x ^= 4;	var x = 6 if (x>5) x = x xor 4

int S=0;
for (int i = 1; i<100;i++)
S += i;
	int S=0;
	for (int i = 1; i<100;i++)
	S += i;
		var S = 0;
		for (i in 1..100)
		S += i;

int x=4;
switch(x)
{
case 1:
printf(“x == 1”);
break;
case 2:
printf(“x == 2”);
break;
default:
printf(“none”);
break;
}
	int x=4;
	switch(x)
	{
	case 1:
	System.out.print(“x == 1”);
	break;
	case 2:
	System.out.print(“x == 2”);
	break;
	default:
	System.out.print(“none”);
	break;
	}
		val x = 4
		when (x)
		{
		1 -> print("x == 1")
		2 -> print("x == 2")
		else -> print("none")
		}

string x = “aaa”, y = “aaa”; if (x.equals(y)) { …… }	String x = “aaa”, y = “aaa”; if (x == y) { …… }	String x = “aaa”, y = “aaa”; if (x === y) { …… }

Tabel 1. Comparație elemente de control simplu al programului între CPP/Java/Kotlin

3. În tabelul doi găsiți o paralelă între clasele/Interfețele principale specifice colecțiilor realizată între Java și Kotlin.

Tip Java	Tip Immutable Kotlin	Tip Mutable Kotlin
Iterator<T>	Iterator<T>	MutableIterator<T>
Iterable<T>	Iterable<T>	MutableIterable<T>
Collection<T>	Collection<T>	MutableCollection<T>
Set<T>	Set<T>	MutableSet<T>
List<T>	List<T>	MutableList<T>
ListIterator<T>	ListIterator<T>	MutableListIterator<T>
Map<K, V>	Map<K, V>	MutableMap<K, V>

Tabel 2. Colecții Java vs Kotlin

4. De un real folos pot fi și așa numitele tabele pentru copiat la examen puse la dispoziție de către academia Kotlin (de unde se poate deduce că este sprijinită adopția de masă la nivel studențesc a limbajului sau trișarea la examen - nu este foarte clar). Tot de aici se trage concluzia că https://blog.kotlin-academy.com s-ar putea să fie o sursă de informații mai utilă decât stackoverflow (Vezi paginile următoare).

Kt. Academy

CHEAT SHEET

BASICS

"Hello, World" program

```
fun main(args: Array<String>) {
    println("Hello, World")
}
```

Declaring function

```
fun sum(a: Int, b: Int): Int {
    return a + b
}
```

Single-expression function

```
fun sum(a: Int, b: Int) = a + b
```

Declaring variables

```
val name = "Marcin"  // Can't be changed
var age = 5          // Can be changed
age++
```

Variables with nullable types

```
var name: String? = null
val length: Int
length = name?.length ?: 0
// length, or 0 if name is null
length = name?.length ?: return
// length, or return when name is null
length = name?.length ?: throw Error()
// length, or throw error when name is null
```

CONTROL STRUCTURES

If as an expression

```
fun bigger(a: Int, b: Int) = if (a > b) a else b
```

For loop

```
val list = listOf("A", "B", "C")
for (element in list) {
    println(element)
}
```

When expression

```
fun numberTypeName(x: Number) = when(x) {
    0 -> "Zero"                 // Equality check
    in 1..4 -> "Four or less"   // Range check
    5, 6, 7 -> "Five to seven"  // Multiple values
    is Byte -> "Byte"           // Type check
    else -> "Some number"
}
```

When expression with predicates

```
fun signAsString(x: Int)= when {
    x < 0 -> "Negative"
    x == 0 -> "Zero"
    else -> "Positive"
}
```

CLASSES

Primary constructor

val declares a read-only property, var a mutable one

```
class Person(val name: String, var age: Int)
// name is read-only, age is mutable
```

Inheritance

```
open class Person(val name: String) {
    open fun hello() = "Hello, I am $name"
    // Final by default so we need open
}
class PolishPerson(name: String) : Person(name) {
    override fun hello() = "Dzień dobry, jestem $name"
}
```

Properties with assessors

```
class Person(var name: String, var surname: String) {
    var fullName: String
        get() = "$name $surname"
        set(value) {
            val (first, rest) = value.split(" ", limit = 2)
            name = first
            surname = rest
        }
}
```

Data classes

```
data class Person(val name: String, var age: Int)
val mike = Person("Mike", 23)
```

Modifier `data` adds:

1. `toString` that displays all primary constructor properties

```
print(mike.toString()) // Person(name=Mike, age=23)
```

2. `equals` that compares all primary constructor properties

```
print(mike == Person("Mike", 23)) // True
print(mike == Person("Mike", 21)) // False
```

3. `hashCode` that is based on all primary constructor properties

```
val hash = mike.hashCode()
print(hash == Person("Mike", 23).hashCode()) // True
print(hash == Person("Mike", 21).hashCode()) // False
```

4. `component1`, `component2` etc. that allows deconstruction

```
val (name, age) = mike
print("$name $age") // Mike 23
```

5. `copy` that returns copy of object with concrete properties changed

```
val jake = mike.copy(name = "Jake")
```

Kt. Academy

CHEAT SHEET

COLLECTION LITERALS

```
listOf(1,2,3,4) // List<Int>
mutableListOf(1,2,3,4) // MutableList<Int>

setOf("A", "B", "C") // Set<String>
mutableSetOf("A", "B", "C") // MutableSet<String>

arrayOf('a', 'b', 'c') // Array<Char>

mapOf(1 to "A", 2 to "B") // Map<Int, String>
mutableMapOf(1 to "A", 2 to "B")
// MutableMap<Int, String>

sequenceOf(4,3,2,1) // Sequence<Int>

1 to "A" // Pair<Int, String>

List(4) { it * 2 } // List<Int>
generateSequence(4) { it + 2 } // Sequence<Int>
```

COLLECTION PROCESSING

```
students
    .filter { it.passing && it.averageGrade > 4.0 }
    // Only passing students
    .sortedByDescending { it.averageGrade }
    // Starting from ones with biggest grades
    .take(10) // Take first 10
    .sortedWith(compareBy({ it.surname }, { it.name }))
    // Sort by surname and then name

generateSequence(0) { it + 1 }
// Infinitive sequence of next numbers starting on 0
        .filter { it % 2 == 0 } // Keep only even
        .map { it * 3 } // Triple every one
        .take(100) // Take first 100
        .average() // Count average
```

Most important functions for collection processing

```
val l = listOf(1,2,3,4)
```

filter - returns only elements matched by predicate

```
l.filter { it % 2 == 0 } // [2, 4]
```

map - returns elements after transformation

```
l.map { it * 2 } // [2, 4, 6, 8]
```

flatMap - returns elements yielded from results of trans.

```
l.flatMap { listOf(it, it + 10) } // [1, 11, 2, 12, 3, 13, 4, 14]
```

fold/reduce - accumulates elements

```
l.fold(0.0) { acc, i -> acc + i } // 10.0
l.reduce { acc, i -> acc * i } // 24
```

forEach/onEach - perfons an action on every element

```
l.forEach { print(it) } // Prints 1234, returns Unit
l.onEach { print(it) } // Prints 1234, returns [1, 2, 3, 4]
```

partition - splits into pair of lists

```
val (even, odd) = l.partition { it % 2 == 0 }
print(even) // [2, 4]
print(odd) // [1, 3]
```

min/max/minBy/maxBy

```
l.min() // 1, possible because we can compare Int
l.minBy { -it } // 4
l.max() // 4, possible because we can compare Int
l.maxBy { -it } // 1
```

first/firstBy

```
l.first() // 1
l.first { it % 2 == 0 } // 2 (first even number)
```

count - count elements matched by predicate

```
l.count { it % 2 == 0 } // 2
```

sorted/sortedBy - returns sorted collection

```
listOf(2,3,1,4).sorted() // [1, 2, 3, 4]
l.sortedBy { it % 2 } // [2, 4, 1, 3]
```

groupBy - group elements on collection by key

```
l.groupBy { it % 2 } // Map: {1=[1, 3], 0=[2, 4]}
```

distinct/distinctBy - returns only unique elements

```
listOf(1,1,2,2).distinct() // [1, 2]
```

Mutable vs immutable collection processing functions

```
val list = mutableListOf(3,4,2,1)
val sortedResult = list.sorted() // Returns sorted
println(sortedResult) // [1, 2, 3, 4]
println(list) // [3, 4, 2, 1]
val sortResult = list.sort() // Sorts mutable collection
println(sortResult) // kotlin.Unit
println(list) // [1, 2, 3, 4]
```

EXTENSION FUNCTIONS TO ANY OBJECT

Reference to receiver \ Returns	Receiver	Results of lambda
it	also	let
this	apply	run/with

```
val dialog = Dialog().apply {
  title = "Dialog title"
  onClick { print("Clicked") }
}
```

Kt. Academy CHEAT SHEET

FUNCTIONS

Function types

`()->Unit` - takes no arguments and returns nothing (`Unit`).
`(Int, Int)->Int` - takes two arguments of type `Int` and returns `Int`.
`(()->Unit)->Int` - takes another function and returns `Int`.
`(Int)->()->Unit` - takes argument of type `Int` and returns function.

Function literals

```
val add: (Int, Int) -> Int = { i, j -> i + j }
// Simple lambda expression

val printAndDouble: (Int) -> Int = {
    println(it)
    // When single parameter, we can reference it using `it`
    it * 2 // In lambda, last expression is returned
}

// Anonymous function alternative
val printAndDoubleFun: (Int) -> Int = fun(i: Int): Int {
    println(i)  // Single argument can't be referenced by `it`
    return i * 2 // Needs return like any function
}

val i = printAndDouble(10) // 10
print(i) // 20
```

Extension functions

```
fun Int.isEven() = this % 2 == 0
print(2.isEven()) // true

fun List<Int>.average() = 1.0 * sum() / size
print(listOf(1, 2, 3, 4).average()) // 2.5
```

DELEGATES

Lazy - calculates value before first usage

```
val i by lazy { print("init "); 10 }
print(i) // Prints: init 10
print(i) // Prints: 10
```

notNull - returns last setted value, or throws error if no value has been set

observable/vetoable - calls function every time value changes. In vetoable function also decides if new value should be set.

```
var name by observable("Unset") { p, old, new ->
    println("${p.name} changed $old -> $new")
}
name = "Marcin"
// Prints: name changed Unset -> Marcin
```

Map/MutableMap - finds value on map by property name

```
val map = mapOf("a" to 10)
val a by map
print(a) // Prints: 10
```

VISIBILITY MODIFIERS

Modifier	Class members	Top-level
Public (default)	Visible everywhere	Visible everywhere
Private	Visible only in the same class	Visible in the same file
Protected	Visible only in the same class and subclasses	Not allowed
Internal	Visible in the same module if class is accessible	Visible in the same module

VARIANCE MODIFIERS