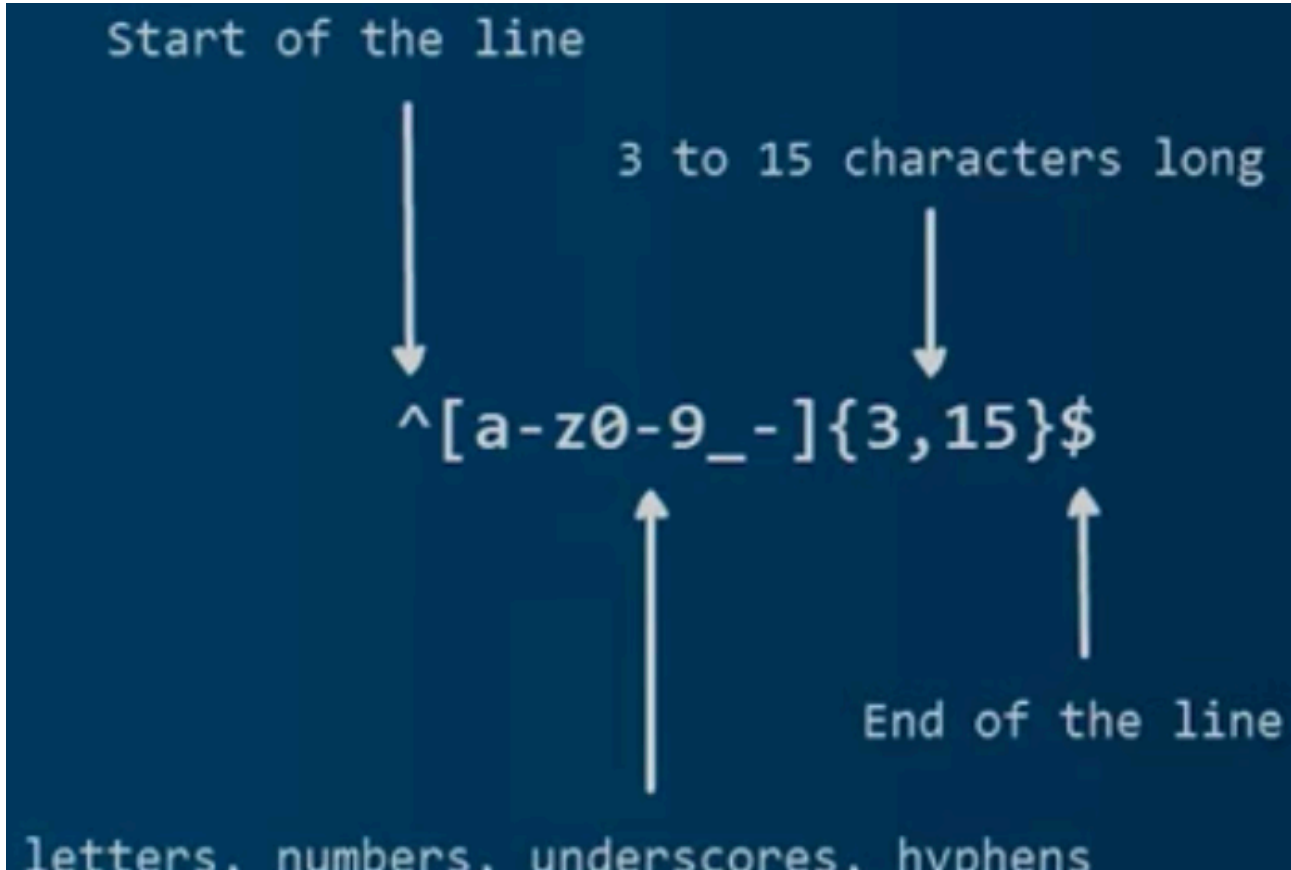
Reguljära uttryck

Regular expressions

- Hjälpmedel för att söka igenom och/eller byta ut text
- Består av sökmönster
- Finns flera verktyg att testa med online, t ex https://www.freeformatter.com/java-regex-tester.html



Java Regex

- Använd för att söka i eller manipulera strängar
- Används ofta för validering av t ex mailadresser

Matcher Class

- boolean matches() – testar om uttrycket matchar mönstret
- boolean find() – letar efter nästa uttryck som matchar mönstret
- boolean find(int start) – letar efter nästa uttryck som matchar mönstret från en viss position
- int start() – det första indexet
- int end() – det sista indexet
- int groupCount() – antalet träffar

Pattern Class

- Static Pattern compile(String regex) – returnerar en instans av ett mönster
- Matcher matcher(charSequence input) – matchar input med mönster
- Static boolean matches(String regex) – kombination av compile och matcher
- String split[] – delar sträng efter matchningar av ett givet mönster
- String pattern() – returnerar mönster
- Int end() – sista indexet

Dags att testa

```
import java.util.regex.*;

public class RegexExample {
  public static void main(String[] args) {
    Pattern pattern = Pattern.compile(".xx."); // . = any character
    Matcher matcher = pattern.matcher("AxxB");
    System.out.println("String matches the given pattern " +
matcher.matches());
  }
}
```

Uppgift

- Utgå ifrån föregående exempel.
- Skriv ett program som matchar alla strängar som innehåller frasen abba och testa uttrycket med ett par strängar. (snabba, sabba, flabba osv)
- Skriv ett mönster som delar upp en sträng på alla `s`.

Regex Character Class

- [abc] – innehåller a, b eller c
- [^abc] – innehåller vad som helst utom a, b eller c
- [a-zA-Z] – range, vilken som helst av bokstäverna a-z eller A-Z.
- [a-d[m-p]] – a-d eller m-p (union)
- [a-z&&[def]] – d, e eller f (intersection)
- [a-z&&[^bc]] – a-z förutom b eller c (subtraction)
- [a-z&&[^m-p]] – a-z men inte m-p (subtraction)
- Samtliga ovanstående matchar för ett tecken.

Dags att testa

```
import java.util.regex.*;

public class CharacterExample {
  public static void main(String[] args) {
    System.out.println(”[xyz]”, ”wbcd”));
    System.out.println(”[xyz]”, ”x”));
    System.out.println(”[xyz]”, ”xxyyyyyz”));
  }
}
```

Uppgift

- Utgå ifrån föregående exempel.
- Skriv ett program som matchar alla strängar som innehåller bokstäverna k-p och testa med ett par strängar.
- Skriv ett mönster som innehåller bokstäverna a-f, men bara om det är i slutet på frasen. (T ex ”spana” skulle ge true men ”banan” skulle inte göra det.)
- Skriv ett mönster som börjar med bokstäverna f-l och inte slutar med z eller y.

Regex Quantifiers

- X? – X förekommer en gång eller inte alls
- X+ – X förekommer mer än en gång
- X* – X förekommer noll eller flera gånger
- X{n} – X förekommer enbart *n* gånger
- X{n,} – X förekommer *n* eller fler gånger
- X{n, m} – X förekommer minst *n* gånger men färre gånger än *m*

Dags att testa

```
import java.util.regex.*;

public class QuantifierExample {
  public static void main(String[] args) {
    System.out.println(Pattern.matches("[ayz]?", "a"));
    // "[ayz]?", "aaa"
    // "[ayz]?", "ayyyyyzz"
    // "[ayz]?", "amnta"
    // "[ayz]?", "ay"
  }
}
```

Dags att testa

```
import java.util.regex.*;

public class QuantifierExample {
  public static void main(String[] args) {
    System.out.println(Pattern.matches("[ayz]+", "a"));
    // "[ayz]+", "aaa"
    // "[ayz]+", "ayyyyyzz"
    // "[ayz]+", "amnta"
    // "[ayz]+", "ay"
  }
}
```

Dags att testa

```
import java.util.regex.*;

public class QuantifierExample {
  public static void main(String[] args) {
    System.out.println(Pattern.matches("[ayz]*", "ayyyyyzza"));
  }
}
```

Uppgift

- Utgå ifrån föregående exempel.
- Skriv ett program som matchar alla s-v minst en gång och testa med ett par strängar.
- Skriv ett mönster som börjar på bokstäverna j-m, men bara om det följs av två förekomster av a-e. (T ex ”jaadå” skulle ge true men ”mala” skulle inte göra det.)

Regex Meta Characters

- . (punkt) – vilket tecken som helst
- d – digit (tal)
- D – non digit
- s – white space (mellanslag, tab, radbrytning)
- S – non white space
- w – word
- W – non word
- b – word boundry
- B – non word boundry

Dags att testa

```
import java.util.regex.*;

public class MetacharExample {
  public static void main(String[] args) {
    System.out.println(”d”, ”abc”));
    System.out.println(”d”, ”1”));
    System.out.println(”d”, ”4443”));
    System.out.println(”d”, ”123abc”));
  }
}
```

Uppgift

- Utgå ifrån föregående exempel.
- Skriv ett program som matchar ett ord följt av två siffror. Använd metachars och testa.
- Skriv om mönstret så att det matchar ord som innehåller minst en förekomst av g-m, exakt ett white space och sedan ett ord som innehåller exakt 2 vokaler.

Scanner

- Scanner kan användas för att läsa indata från filer, tangentbord osv.

```
import java.util.Scanner;

public class GuessNumber {
  public static void main(String[] args) {
    try (Scanner sc = new Scanner(System.in))
    {
      System.out.println(”Skriv ett tal”);
      String guess = sc.nextLine();
      System.out.println(”Ditt tal: ” + guess);
    }
  }
}
```

Uppgift

- Utgå ifrån föregående exempel.
- Ta reda på hur man slumpar fram ett tal från 1-100.
- Låt användaren gissa vilket tal du har slumpat fram och skriv ut ”Högre!” eller ”Lägre” om användaren gissar fel, ”Grattis, du gissade rätt på 4 gissningar!” eller hur många gissningar som nu krävdes.
- Lägg till felhantering genom att fånga upp eventuella fel. Försök ”krascha” programmet genom att ge input det inte förväntar sig. Se till att ta hand om felet.

Serialization

Serialisering

- Har ni någonsin tänkt att det är lite tröttsamt att hålla på och skapa upp massa objekt hela tiden?
- Skulle det inte vara trevligt att kunna läsa in objekten direkt från en fil?
- Och det kan man
- Med hjälp av serialisering och deserialisering

Serialisering

- Serialisering innebär att ett objekt ”plattas ut” och sparas ner till en fil.
  - Inte helt olikt JSON.parse() och JSON.stringify() i JavaScript.
- Filer med serialiserat data har suffixet `.ser`
- `.ser`-filer kan skrivas mha ObjectOutputStream.writeObject
- De objekt som serialiseras måste implementera interfacet Serializable
- Många av Javas standardklasser implementerar Serializable
- Funkar både för enskilda objekt och för samlingar (t.ex Listor)

Deserialisering

- Deserialisring innebär att ett objekt läses upp från en `.ser`-fil
- `.ser`-filer läses mha ObjectInputStream.readObject
- Casta det upplästa objekten till rätt klass.

Exempel serialisering

```
import java.io.*;

class Persist {
 public static void main(String[] args) throws Exception {
  Student s1 = new Student(211, ”Kalle");

  ObjectOutputStream out =
       new ObjectOutputStream(FileOutputStream("f.txt");

  out.writeObject(s1);
  out.flush();
  System.out.println("success");
 }
}
```

Exempel deserialisering

```
import java.io.*;

class Depersist{
 public static void main(String[] args) throws Exception {

  ObjectInputStream in =
       new ObjectInputStream(new FileInputStream("f.txt");
  Student s = (Student) in.readObject();
  System.out.println(s.id + " ” + s.name);

  in.close();
 }
}
```

Serialisering

- I vårt exempel skriver vi och läser ifrån en fil
- Vanligare att skicka serialiserade objekt mellan program
- Används mycket i client-server arkitekturer
  - Klienten ber servern att skicka över data
  - Servern serialisrear de objekt klienten ber om och skickar dem via en socket
  - Klienten tar emot objekten, deserialiserar dem och använder.

Uppgift

- Skapa ett program med klassen Account.
  - Ska hålla reda på kontots ägare och saldo.
- Skapa 4-5 konton och skapa metoder som sparar och hämtar kontona till filer.
- Extrauppgifter:
  - Lägg till klassen Person och låt en Person vara ägare till ett konto.
  - Implementera metoder för att föra över pengar mellan konton.

Inlämningsuppgift

- Det saknas fortfarande några detaljer, men vi ska bygga en applikation som läser in frågor till ett Quiz från en fil med hjälp av strömmar.
- Vi ska bygga något frågesport-liknande där man på samma dator får turas om mellan två spelare att svara på frågor.
- Spelet kommer att gå på tid, man ska bara ha en viss tid på sig att svara på frågorna.
- Applikationen kommer att använda flera trådar.
- Applikationen kommer att kräva att ni implementerar abstrakta klasser och/eller interface.
- Ni ska använda något/några design patterns som vi kommer att prata om senare.
- Inget fokus på gränssnitt.