Sovelto

Java 8 uusia piirteitä

Java 8 uusia piirteitä

- Java 8 DateTime API
  - LocalDate, LocalTime, …
  - Period, Duration
  - https://docs.oracle.com/javase/8/docs/api/java/time/package-summary.html

- Interface'ien muutokset:
  - default metodi
  - static metodit
  - Funktionalliset rajapinnat (Functional interfaces)
    - https://docs.oracle.com/javase/8/docs/api/java/util/function/package-summary.html

- Streamit
- Optional

Stream API

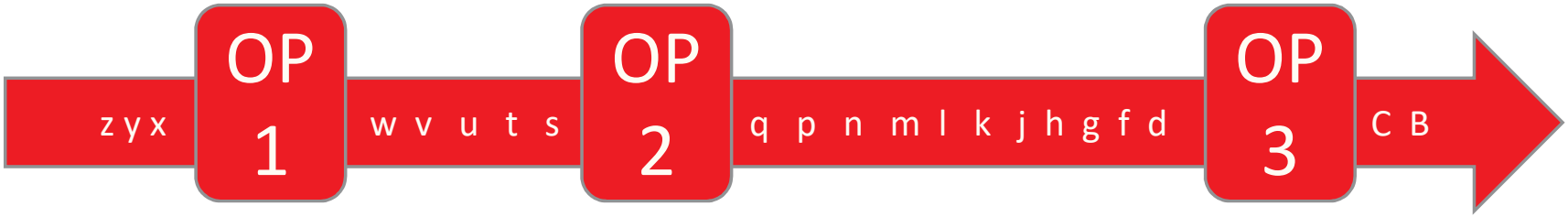
Java 8

Sovelto

Java 8 Stream API

- Java 8 hakemistolistaus onnistuu helposti Files luokan avulla, mutta tuloksena onkin Stream<Path> tai DirectoryStream
- Näiden läpikäynti vaatii perustiedot Stream APIsta
- Perustana rajapinta java.util.Stream
  - Se osaa genericsin
  - Joitain erityisiä Streameja löytyy myös, esimerkiksi primitiivityypeille erikoisvirrat kuten IntStream
  - Kokoelmista saa aina streamin stream() metodilla, taulukoista Arrays-luokan avulla
- Streamin luomisen/löytämisen jälkeen alkiot liikkuvat sitä virtaa pitkin, ja niille voidaan tehdä operaatioita joko sarjallisesti tai rinnakkain
  - Operaatiot eivät siis odota että kaikki streamin alkiot ovat saapuneet, vaan käsittelee niitä tyypillisesti yksi kerrallaan ja lähettää käsitellyn virtaan eteenpäin

Esimerkki




```
String aakkoset =
"abcdefghijklmnopqrstuvwxyz";
IntStream virta = aakkoset.chars();
String tulos = virta
        .mapToObj(i->""+(char)i)
        .filter(c -> !"aeiouy".contains(c))
        .map(String::toUpperCase)
        .collect(Collectors.joining(" "));
System.out.println(tulos);
```

Perustoiminta

- Virran luonti esim. kokoelmilla: stream()
- Toiminnallisuus - virta jatkuu
  - filter - filtteröi virtaa
  - map - muokkaa alkioita
  - sorted - järjestetty
- Koonti - virta loppuu
  - forEach - tee jokaiselle alkiolle
  - collect - geneerinen, esim. Collectors luokan toList(),groupingBy(), ja joining()
  - Virtatyyppispesifejä, esim. IntStreamilla max, min, average yms.

- Stream tyypin vaihtaminen
  - primitiivi -> olio: mapToObject
  - olio -> primitiivi: mapToInt, mapTo
  - olio -> olio: map

Funktiot

- Lambdat, eli nimettömät metodit. Lyhyitä, virtojen kanssa tyypillisesti vain yksi parametri, ja lyhyt runko implisiittisellä returnilla. Tyypit päätellään
  - c -> c * 2
  - (a, b) -> a + b
  - (c) -> {return c>>1;}
  - () -> rnd.nextInt();
- Funktioreferenssi
  - String::toUpperCase
    - Sama kuin kirjoittaisi lambdan: s -> s.toUpperCase()
- Funktiotyypit, yleisimmät java.util.function paketista:
  - https://docs.oracle.com/javase/8/docs/api/java/util/function/package-summary.html
  - Function<T, R> Funktio joka ottaa yhden parametrin ja palauttaa arvon T input-tyyppi, R result-tyyppi, esim. String joku(Integer)
  - Supplier<T> - Tuottaa arvon ilman parametria
  - Consumer<T> - Ottaa yhden parametrin, mutta ei palauta mitään
  - Predicate<T> - Ottaa yhden parametrin ja palauttaa booleanin

Esimerkkejä

- Funktiotyyppi ja lambda yhdistelmänä

```
public class MerkkijonojenKertomaEsimerkki {
    private static BiFunction<Integer, String, String> kertaa =
            (lkm, m) -> new String(new char[lkm]).replace("\0", m);
    public static void main(String[] args) {
        System.out.println(kertaa.apply(3, "x")); // xxx
        System.out.println(kertaa.apply(7, "i")); // iiiiiii
    }
}
```

- Henkilöiden käsittelyä

```
public void tulostaHenkilot(List<Henkilo> lista) {
    lista.stream().forEach(System.out::println);
    // tai näin:
    String hlot = lista.stream()
            .map(h -> h.toString())
            .collect(Collectors.joining(" | "));
    System.out.println(hlot);
}
```

```
public int ikienSumma(List<Henkilo> lista) {
    int summa = lista.stream()
            .mapToInt(Henkilo::getIka)
            .sum();
    return summa;
}
```

Stream APIn käyttö

- Etsi .java tiedostot tietystä hakemistosta

```
Files.list(polku)
        .filter(Files::isRegularFile)
        .filter(p->p.getFileName().toString().endsWith(".java"))
        .forEach(System.out::println);
```

- Etsi .java tiedostot Työhakemiston alta **ja** listaa tiedoston sanat

```
Path cwd = Paths.get(".");
BiPredicate<Path, BasicFileAttributes>  hakufunktio =
        (p, attr) -> Files.isRegularFile(p) && p.getFileName().toString().endsWith(".java");
Stream<Path> tiedostot = Files.find(cwd, 10, hakufunktio);
tiedostot.forEach(System.out::println);
```

```
Path main = Paths.get("./src").resolve("fi/academy/Main.java");
String kaikki = Files.lines(main)
        .map(rivi -> rivi.split("[\\s\\p{Punct}]+")) // Stream<String[]>
        .flatMap(Arrays::stream)                     // Stream<String>
        .distinct()
        .collect(Collectors.joining(" "));
System.out.println(kaikki);
```

Hieman lisää esimerkkejä

- Kymmenen satunnaista lukua väliltä 0-999

```
final Random rnd = new Random();
List<Integer> numerot = IntStream
        .generate(()->rnd.nextInt(1000))
        .limit(10)
        .mapToObj(Integer::valueOf)
        .collect(Collectors.toList());
```

- Hieman laajennettu versio edellisen sivun esimerkistä, mitä tämä tekee?

```
Path main = Paths.get("./src").resolve("Sanalaskuri.java");
Map<String, Long> kaikki = Files.lines(main)
        .map(rivi -> rivi.split("[\\s\\p{Punct}]+"))        // Stream<String[]>
        .flatMap(Arrays::stream)                            // Stream<String>
        .filter(s -> !s.isEmpty())                          // tyhjät rivit pois
        .collect(Collectors.groupingBy(Function.identity(), Collectors.counting()));
kaikki.entrySet().stream()
        .sorted(Map.Entry.<String, Long>comparingByValue().reversed())
        .forEachOrdered(e -> System.out.println(String.format("%2$4d: %1$s",
                e.getKey(), e.getValue())));
```

Primitiivivirrat

- Virrat käyttävät genericsiä, eli virran alkioilla on aina tyyppi
  - Stream<Integer> tms.
- Primitiivityyppejä, int/char/boolean, ei voi käyttää genericsin kanssa. Virtoja halutaan kuitenkin usein käyttää myös niiden kanssa
- Primitiivityypeille onkin omat luokat, IntStream, DoubleStream jne.
  - Oikeammin rajapinnat, esim.

    public interface **IntStream** extends BaseStream<Integer,IntStream>
- Primitiivityyppien virroilla voikin olla näin omia metodeita juuri kyseisen tyyppisille alkioille
- IntStream: int sum(), OptionalDouble average(), OptionalInt max()

Optional

- Optional on luokka, joka voi sisältää arvon, tai voi olla sisältämättä

```
Optional<String> optional = Optional.empty();
System.out.println(optional.orElse("Oli tyhjä")); // oli tyhjä
optional = Optional.of("Arvo");
System.out.println(optional.orElse("Oli tyhjä")); // Arvo
System.out.println(optional.get());               // Arvo
```

- Sitä käytetään null arvon palauttamisen sijaan, jotta null *mahdollisena* paluuarvona tulisi paremmin huomioitua
- Primitiivityypeille on omat OptionalInt, OptionalDouble jne. tyypit
- Eli nyt voidaan palauttaa ikään kuin null vaikka paluutyyppinä olisikin primitiivityyppi
  - Esimerkiksi IntStream.average() käyttää tätä, eli jos IntStream on tyhjä, niin average palauttaa OptionalInt olion, jolle isPresent() on false