Java 8 New Features

Sunit Parekh
Sarthak Makhija

Java 8 Features

- Lambda
- Method references
- Streams
- Date/Time API
- Default Methods
- Optional
- Nashorn, JavaScript engine

and more...

Lambda Expression `() -> {....}`

- Declaring the types of the parameters is optional.
- Using parentheses around the parameter is optional for only one parameter.
- Using curly braces is optional (unless you need multiple statements).
- The “return” keyword is optional if you have a single expression.

Lambda Expression

- `() -> System.out.println(this)`
- `(String str) -> System.out.println(str)`
- `str -> System.out.println(str)`

- `(String s1, String s2)-> {return s2.length() - s1.length();}`
- `(s1, s2) -> s2.length() - s1.length()`

Lambda: Examples

```
Arrays.sort(strArray,
            (String s1, String s2) -> s2.length() - s1.length() );
```

```
String sql = "delete * from User";
getHibernateTemplate()
.execute(session -> session.createSQLQuery(sql).uniqueResult());
```

Lambda: Scope

```
1  import static java.lang.System.out;
2
3  public class Hello {
4      Runnable r1 = () -> out.println(this);
5      Runnable r2 = () -> out.println(toString());
6
7      public String toString() { return "Hello, world!"; }
8
9      public static void main(String... args) {
10         new Hello().r1.run(); //Hello, world!
11         new Hello().r2.run(); //Hello, world!
12     }
13 }
```

Lambda: Method Reference

```
Files.lines(Paths.get("Nio.java"))
             .map(String::trim)
             .forEach(System.out::println);
```

Method references can point to:

- Static methods.
- Instance methods.
- Methods on particular instances.
- Constructors (ie. TreeSet::new)

Lambda Live Coding

Streams

- map
- filter
- peek
- limit
- parallelStream

Streams: map

```
class Student {
      public String name;
      public LocalDate birthDate;
}
List<Student> students = ...

Stream<String> names = students.stream().map( s -> s.name );
List<String> namesAsList = names.collect(Collectors.toList());
```

Streams: map

```
class Student {
      public String name;
      public LocalDate birthDate;
}
List<Student> students = ...

List<String> names = students.stream()
                    .map( s -> s.name )
                    .collect(Collectors.toList());
```

Streams: filter

```
class Student {
      public String name;
      public LocalDate birthDate;
}
List<Student> students = ...

List<Student> leapYearStudents = students.stream()
                .filter(s -> s.birthDate.isLeapYear())
                .collect(Collectors.toList());
```

Streams: peek & limit

```
class Student {
      public String name;
      public LocalDate birthDate;
}
List<Student> students = ...
List<String> names = students.stream()
                .map( s -> s.name)
                .limit(5)
                .peek(System.out::println)
                .collect(Collectors.toList());
```

Streams: parallelStream

```
class Student {
      public String name;
      public LocalDate birthDate;
}
List<Student> students = ...

List<Student> leapYearStudents = students.parallelStream()
                .filter(s -> s.birthDate.isLeapYear())
                .collect(Collectors.toList());
```

Collectors

```
// Accumulate names into a List
List<String> list = people.stream().map(Person::getName).collect(Collectors.toList());

// Accumulate names into a TreeSet
Set<String> set =
people.stream().map(Person::getName).collect(Collectors.toCollection(TreeSet::new));

// Convert elements to strings and concatenate them, separated by commas
String joined = things.stream().map(Object::toString).collect(Collectors.joining(", "));

// Compute sum of salaries of employee
int total = employees.stream().collect(Collectors.summingInt(Employee::getSalary)));
```

Streams Live Coding

New DateTime API (java.time)

```
LocalTime now = LocalTime.now();
LocalTime later = now.plus(8, HOURS);

LocalDate today = LocalDate.now();
LocalDate date = LocalDate.of(2015,12,12); // (yyyy,MM,dd)
LocalDate thirtyDaysFromNow = today.plusDays(30);
LocalDate nextMonth = today.plusMonths(1);
LocalDate aMonthAgo = today.minusMonths(1);

LocalDateTime now = LocalDateTime.now();
```

NULLS - NPE - Friend For Life

Null references are a source of too many problems.

Alternatives to avoid NPE -

- Perform null checks
- Eagerly instantiate an object

Being Defensive - NULL checks

```
public String getInsuranceCompanyName (Car car){
    if ( car != null ){
        if ( car.getInsurance() != null ) {
            return car.getInsurance().getName();
        }
        return “Unknown”;
    }
    return “Unknown”;
}
```

Ugly, nested with null checks,complexity increases with deep object graph

Optional

Java SE 8 introduces a new class called java.util.Optional<T>. E.g;

```
public class Car {
    private Optional<Insurance> insurance;
    public Car(Optional<Insurance> insurance){
        this.insurance = insurance;
    }
    public Optional<Insurance> insurance(){
        return insurance;
    }
}
```

Optional

```
public class Insurance {
    private String name;
    public Insurance ( String name ){
        this.name = name;
    }
    public String insuranceName(){
        return name;
    }
}
```

Optional

```
public String getInsuranceCompanyName (Optional<Car> car){
    return    car.flatMap(Car::insurance)
                  .map(Insurance::insuranceName)
                  .orElse(“Unknown”);
}
```

Future Reading

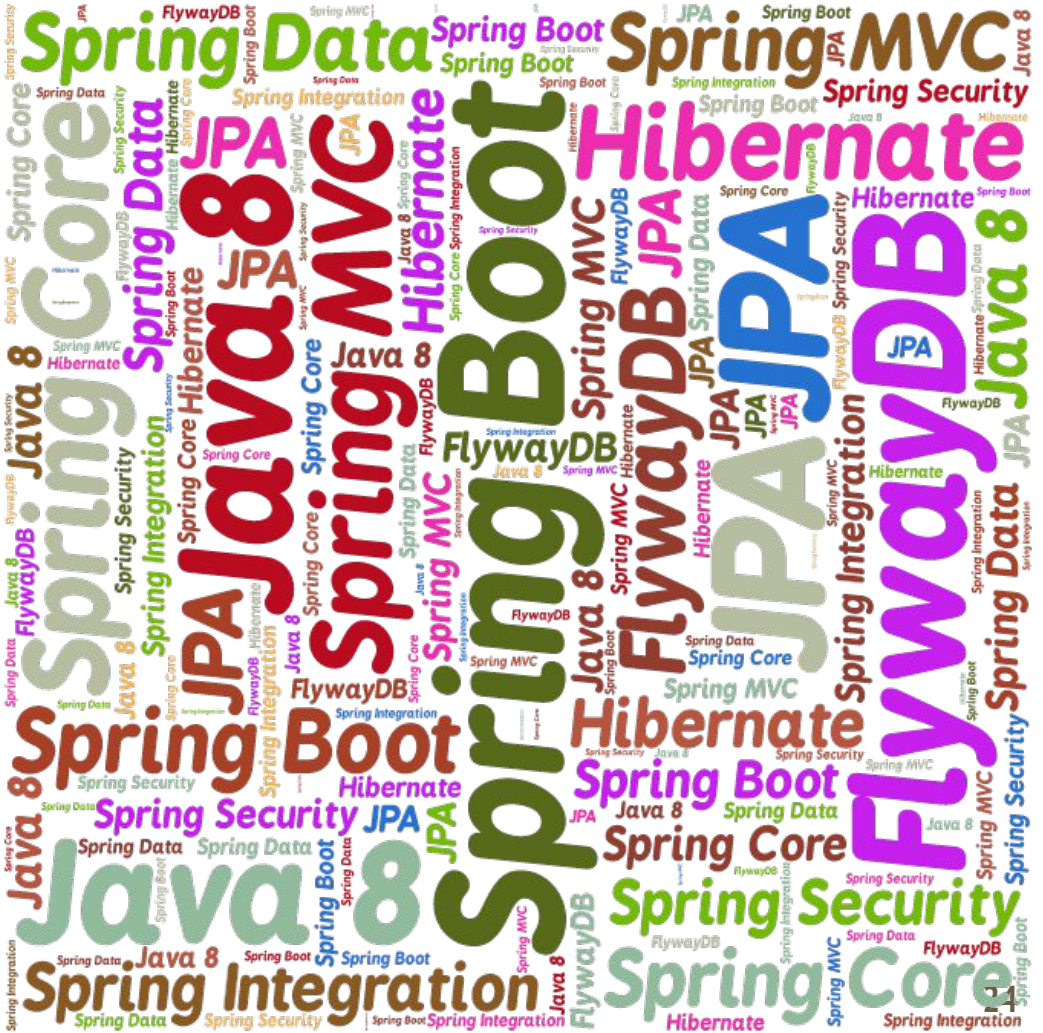
- https://leanpub.com/whatsnewinjava8/read
- http://winterbe.com/posts/2014/03/16/java-8-tutorial/
- http://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/
- https://www.youtube.com/playlist?list=PLSM8fkP9ppPoiRtiyZA9ryXSg6LtyNb-3

Thanks!

Like to reach us,

Sunit Parekh
parekh.sunit@gmail.com

Sarthak Makhija
sarthak.makhija@gmail.com



Grouping By

```
Stream<Movie> movies = ...

movies.collect(Collectors.groupingBy(Movie::getGenre, Collectors.counting()));

movies.collect(
     Collectors.groupingBy(Movie::getLeadActor,
                           Collectors.summingDouble(Movie::getCollectionInCrore)
     ));

movies.collect(
     Collectors.groupingBy(Movie::getReleaseYear,
                           Collectors.mapping(Movie::getTitle, Collectors.toList())
     ));
```

Partitioning By

```
Stream<Movie> movies = ...

Map<Boolean, List<Student>> oldAndNewMovies =
          movies.collect(
                    Collectors.partitioningBy(m -> m.getReleaseYear() > 2010)
          );
```