New features in Java 8:

1. Lambda Expression

Allows me to write a method in the same place you are going to use it. It is a method without declration

* 1. grouping a collection by the value of a specific class

Map<Person.Sex, List<Person>> byGender = rosterList.stream().collect(Collectors.groupingBy(Person::getGender));

1. Parallel operation

ConcurrentMap<Person.Sex, List<Person>> byGender = roster.parallelStream().collect(Collectors.groupingByConcurrent(Person::getGender))

Using common- fork- pool to parallel the iteration

1. New date time API

Instant – represents a point in time (timestamp)

LocalDate – represents a date (year, month, day)

LocalDateTime – same as LocalDate, but includes time with nanosecond precision

OffsetDateTime – same as LocalDateTime, but with time zone offset

LocalTime – time with nanosecond precision and without date information

ZonedDateTime – same as OffsetDateTime, but includes a time zone ID

OffsetLocalTime – same as LocalTime, but with time zone offset

MonthDay – month and day, without year or time

YearMonth – month and year, without day or time

Duration – amount of time represented in seconds, minutes and hours. Has nanosecond precision

Period – amount of time represented in days, months and years

**Optionals**

It is a container object which is used to contain not-null objects. isPresent()

Optioanls opt = Opetional.ofNullable(value)

opt.isPresent()

**stream** ()

gets/computes elements on demand. It never stores the elements.

Source − Stream takes Collections, Arrays, or I/O resources as input source.

Aggregate operations − Stream supports aggregate operations like filter, map, limit, reduce, find, match, and so on.

1. Concurrent accumulators

Updating of numeric counters accessed by multiple threads

Java 8 offers atomic package

AtomicInteger atomicInt = new AtomicInteger(0);

<http://winterbe.com/posts/2015/05/22/java8-concurrency-tutorial-atomic-concurrent-map-examples/>

LongBinaryOperator op = (x, y) -> 2 \* x + y;

LongAccumulator accumulator = new LongAccumulator(op, 1L);

ExecutorService executor = Executors.newFixedThreadPool(2);  
IntStream.range(0, 10).forEach(i -> executor.submit(() -> accumulator.accumulate(i)));

stop(executor);

System.out.println(accumulator.getThenReset()); // => 2539

Memory leak :

1. Java heap leaks :
   1. Static fields holding on the object reference

Heavy size static map = New HashMap<1000000>();

* 1. Calling String.inter() on very long string object

It runs in permGen space of memory so then adding memory to PermGen would be 1 solution

* 1. Forgetting to close Stream
  2. Forgetting to close Connection
  3. Adding object in to HashSet without having equal() and hashCode()

In the class object