Java 8 features cheat sheet

Lambdas

Based on *@FunctionalInterface*

```
(parameters) -> expression
(parameters) -> {statements;}
```

Examples: () -> {} | ()->”Henry”
(Car c)-> { return Car::Engine;} (x)->x*x

Predicate<T>	T -> Boolean
Consumer<T>	T-> void
Supplier<T>	() ->T
Function<T,R>	T-> R
UnaryOperator<T>	T->T
BinaryOperator<T>	(T,T) -> T
BiPredicate<L,R>	(L,R) -> boolean
BiConsumer<L,R>	(L,R) -> void
BiFunction<L,R,U>	(L,R)->U

Method Reference

1. Static method Integer::parseInt
2. Instance method String::length
3. Existing object car::getEngine

Example
stringList.sort(String::compareToIgnoreCase)
Supplier<Car> c1 = Car::new; Car c = c1.get()
Function<Int..,Car> c2=Car::new; c=c2.apply(10)

Comparator.comparing(Car::getWeight).reversed().thenComparing(Car::getModel)

Streams

Stream is a set of values spread out in time, collection is set of values spread out in space

filter	I	Stream<T>	Predicate<T>
distinct	I	Stream<T>	
skip	I	Stream<T>	long
limit	I	Stream<T>	Long
map	I	Stream<R>	Function<T,R>
flatMap	I	Stream<R>	Function<T, Stream<R>>
sorted	I	Stream<T>	Comparator<T>
anyMatch, noneMatch, allMatch,	T	Bool	Predicate<T>
findAny, findFirst	T	Optional<T>	
forEach	T	Void	Consumer<T>
collect	T	R	Collector<T,A,R>
reduce	T	Optional<T>	BinaryOperator<T>
count	T	Long	
iterate	I	Void	Stream<T>
generate	I	Void	Stream<T>

Examples:
transactions.stream().anyMatch(transaction->transaction.getTrader().getCity().equals(“LA”);
tr.stream().map(Transaction::getValue).reduce(Integer::sum)=tr.stream().toIntMap(Transaction::getValue).sum();
tr.stream().map(Transaction::getTrader).filter(trader->trader.getCity().equals(“LA”).distinct().sorted(comparing(Trader::getName)).collect(toList());
menu.stream().map(Dish::getName).collect(joining(“, “));
menu.stream().collect(groupingBy(Dish::getType,groupingBy(Dish::getSpycines));
menu.stream().partitionBy(Dish::isVegeterian,groupBy(Dish::getType));

Collectors:

toList, toSet, toCollection
(toCollection(),ArrayList::new)
counting,summingInt,averagingInt,summurizingInt(summingInt(Dish::getCalories())
joining (joining(“, “))
maxBy,minBy(collect(minBy(comparingInt(Dish::getCalories)))
reducing
(reducing(0,Dish::getCalories,Integer::sum)
collectingAndThen,groupingBy,partitioningBy

Default methods

Needed to evolve API

Example:
default List<T> sort(List<T> l){
Collections.sort(l);}

Optional

Handle null values better, declare that method could return a null value.

Optional.empty(); Optional.of(o); Optional.ofNullabe(o)

Examples:
optoinalP.flatMap(Person:getCar).flatMap(Car::getEngine).map(Engine::getCylinders).orElse(value);

optionalC.ifPresnet(); optionalC.get(); optionalC.orElseThrow(e)

Completable Future

Make it easy to work with futures and parallel computations.

$N_{trheads}=N_{cpu}*U_{cpu}(1+W/C)$ where N_{cpu} is number of core CPU (Runtime.getRuntime().availableProcessors()) U_{cpu} is target CPU utilization between 0 and 1 and W/C is ration of wait to compute time. If there is I/O and few compute statements ratio is 100 since wait is dominant.

supplyAsync	Asynchronous task to get executed. You can pass executor as a second parameter
thenApply	Dependent computation to be performed on the same thread as prior computation
thenCompose	Dependent computation to be applied on new thread
thenCombine	Merging current parallel computation with another parallel computation define as a second parameter. Third parameter describes how to combine the computations
thenAccept	Take a consumer and applies it to the computed result performed on the same thread.
thenAcceptAsync	Same as thenAccept done asynchronously

Examples:
shops.stream().map(shop -> CompletableFuture.supplyAsync(() -> shop.getPrice(product),executor)).map(future -> future.thenApply(Quote::parse)).map(future -> future.thenCompose(quote-> CompletableFuture.sypplyAsync(() -> Discount.applyDiscount(quoute), executor)));

Date and Time API

Main purpose is to overcome the limitation of existing date APIs

Classes (inspired by JIDE package):

LocalDate, LocalTime, LocalDateTime, Instant, Duration, Period.

Date manipulations are done by TemporalAdjuster functional interface.