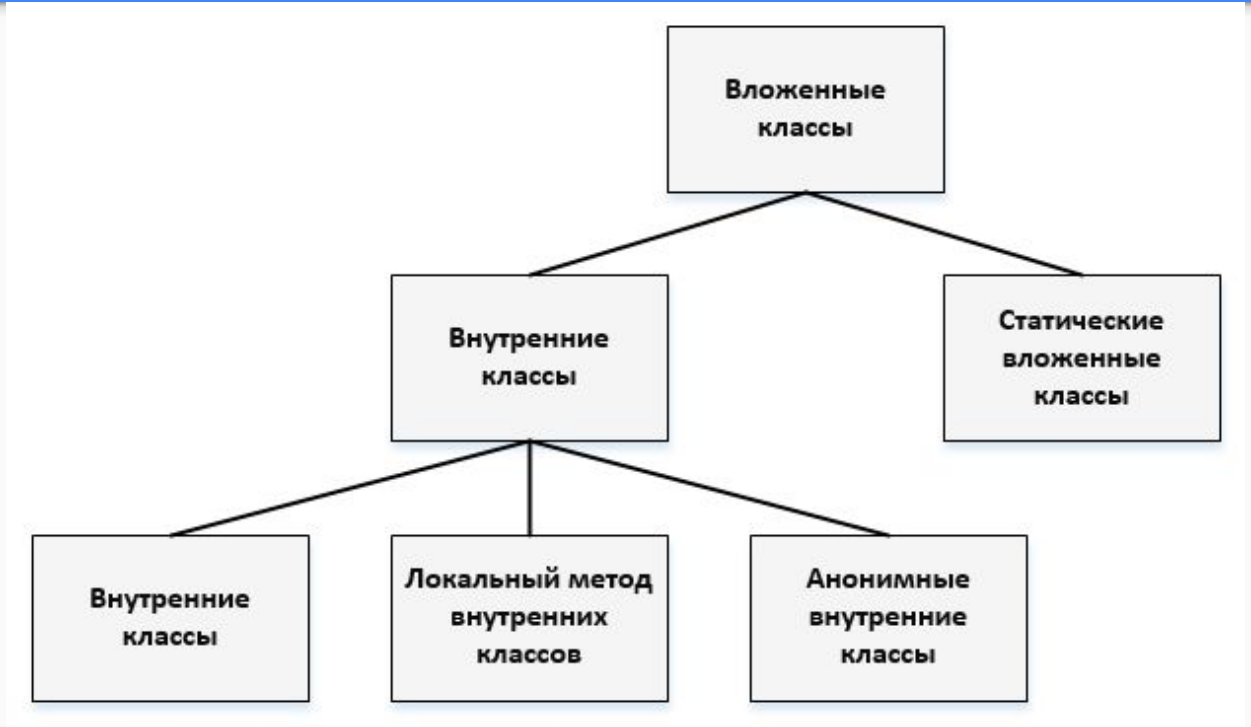
Java

Вложенные классы, функциональные интерфейсы

Вложенные классы в Java

- локальные классы
- статические и нестатические классы
- затенение
- анонимные классы

Вложенные классы в Java



Статические и нестатические вложенные классы

```
class OuterClass {
    ...
    static class StaticNestedClass { ... }
    class NestedClass { ... }
}
```

Зачем нужны вложенные классы

- Уменьшение количества однообразного кода
- Уменьшение сущностей может упростить чтение кода (Runnable)
- Добавление связанности сущностей (Map.Entry)

Статические классы

- Могут использоваться отдельно от основного класса
- Ничем не отличаются от обычных

Локальные классы

- Используются для связи с сущностью
- создание - myClassInstance.new MyInnerClass();
- Не может существовать без внешнего класса
- Не может использоваться в статических методах внешнего класса
- Имеет доступ к внутренностям внешнего класса
- Могут существовать в рамках метода
- Используется редко

Анонимные классы

- Используется для быстрой реализации по месту
- Обычно реализуются небольшие интерфейсы или классы
- Не имеют имени, но имеют базовый тип или интерфейс
- Имеют доступ к текущему “скоупу”

Функциональные интерфейсы

```
@FunctionalInterface
public interface Comparator<T> {
    int compare(T var1, T var2);
    ...
}
```

Функциональные интерфейсы (Comparator)

```
new TreeSet<>(new Comparator<T>() {
    @override
    int compare(T var0, var1) {
        …
    }
}
```

Стандартные функциональные интерфейсы

- Comparator<T>
- Consumer<T>
- Function<T, R>
- Predicate<T>
- Supplier<T>
- Bi - формы
- Пакет java.util.function

Predicate<T>

- boolean test(T var1);
- Проверка соблюдения некоторого условия
- Если оно выполняется то вернётся true

Function<T, R>

- R apply(T var1);
- Функция перехода типа T к типу R

Consumer<T>

- void accept(T var1);
- Совершает некоторое действие над объектом типа T

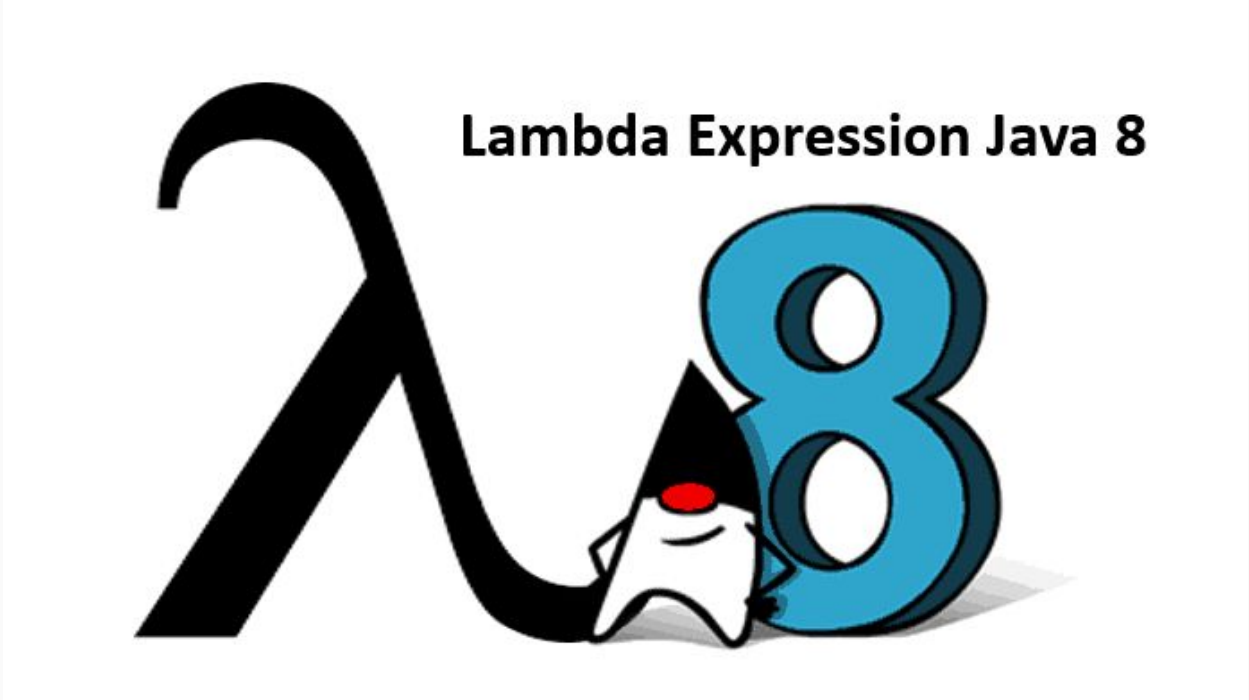
Supplier<T>

- T get();
- Ничего не принимает

Зачем это всё?

Правда, зачем?

Лямбда-выражения!



Лямбда-выражения

Пример: Comparator

Краткий формат лямбд

- (x, y) -> x + y
- Типы можно опустить

Список аргументов	Значок стрелки	тело
`(int x, int y)`	`->`	`x + y`

Связь с функциональными интерфейсами

- Comparator<T> => lambda

Форматы записи

- No arguments: `() -> System.out.println("Hello")`
- One argument: `s -> System.out.println(s)`
- Two arguments: `(x, y) -> x + y`
- With explicit argument types:

```
(Integer x, Integer y) -> x + y
```

- Multiple statements:

```
(x, y) -> {
    System.out.println(x);
    System.out.println(y);
    return (x + y);
}
```

Ссылки на методы

Method reference

1. `HashFunc::compute1`

   `(x) -> HashFunc::compute1(x)`

2. `HashFunc::compute2`

   `(x, y) -> HashFunc::compute2(x, y)`

```
public class HashFunction {
    public static int compute1(Object obj) { /*...*/ }
    public static int compute2(int a, int b) { /*...*/ }
}
```

Runnable тоже может быть лямбдой

- new Thread(new Runnable() { … })
  - void run();
- new Thread(() -> …)