Lambda 表达式

-> 和 ::

->

- (int x, int y) -> {return x + y;}
- (x, y) -> {return x + y;}
- (x, y) -> x + y
- (x) -> {return x * x;}
- x -> x * x
- () -> {return 5;}

  () -> 5

(arg1, arg2, …) -> { body }

:: eta转换

- String::valueOf // 引用String类的静态方法valueOf
- System.out::print // 引用System静态属性out 的print方法
- obj::toString // 实例obj的 toString方法
- super::methodName // 当前类的父类的 methodName 方法
- ArrayList::new // 对象ArrayList的构造函数，用于创建ArrayList对象
- TypeName[]::new // 数组的创建

Functional interfaces

- 接口，不能是抽象类
- 有且只有一个抽象方法
- 可以有0到多个default方法
- 也可以有0到多个静态方法
- 可以使用 @FunctionalInterface 注解，协助语法检查

```
public interface MyFunction{
    int cal(int x, int y);
}
```

```
@FunctionalInterface
public interface MyFunction2{
    int cal(int x, int y);

    default int add(int x, int y){
        return x + y;
    }

    static int multiple(int x, int y){
        return x * y;
    }
}
```

```java
public interface MyFunction{
    int cal(int x, int y);
}
```

```java
MyFunction mf1 = (x, y) -> {return x + y;};
MyFunction mf2 = (x, y) -> x + y;

int r = mf1.cal(5, 10);
```

```java
MyFunction mf3 = Math::addExact;
int r2 = mf3.cal(5, 10);
```

- java.lang.Runnable
- java.awt.event.ActionListener
- java.util.Comparator
- java.util.concurrent.Callable
- java.util.funciton
  - Function<T, R> 函数型，接收一个输入参数T，返回结果R
  - Consumer<T> 消费型，接收一个输入参数T，无返回结果
  - Predicate<T> 断言型，接收一个输入参数T，返回一个boolean型结果
  - Supplier<T> 供给型，无输入参数，返回结果T

```
List<Employee> list;

list.sort(new Comparator<Employee>() {
    @Override
    public int compare(Employee e1, Employee e2) {
        return e1.getName().compareTo(e2.getName());
    }
});
```

```
list.sort((Employee e1, Employee e2) -> {
       return e1.getName().compareTo(e2.getName());
    });
```

```
list.sort((e1, e2) -> e1.getName().compareTo(e2.getName()));
```

`static <T,U extends Comparable<? super U>> Comparator<T> comparing(Function<? super T,? extends U> keyExtractor)`	Accepts a function that extracts a `Comparable` sort key from a type `T`, and returns a `Comparator<T>` that compares by that sort key.

```
list.sort(Comparator.comparing(
                    (Employee employee) -> { return employee.getName();}
                )
        );
```

```
list.sort(Comparator.comparing(employee -> employee.getName()));
```

```
list.sort(Comparator.comparing(Employee::getName));
```

```
import static java.util.Comparator.comparing;

list.sort(comparing(Employee::getName));
```