



Немного «примеров»


Что выведет следующий код?

```
List<String> list = new ArrayList<>();  
list.add("молоко");  
list.add("хлеб");  
list.add("колбаса");  
Stream<String> stream = list.stream();  
list.add("яйца, яйца ещё!");  
stream.forEach(System.out::println);
```

- A. молоко/хлеб/колбаса
- B. молоко/хлеб/колбаса/яйца, яйца ещё!
- C. молоко/хлеб/колбаса/ConcurrentModificationException
- D. ConcurrentModificationException

Что выведет следующий код?

```
List<String> list = new ArrayList<>();  
list.add("молоко");  
list.add("хлеб");  
list.add("колбаса");  
Stream<String> stream = list.stream();  
list.add("яйца, яйца ещё!");  
stream.forEach(System.out::println);
```

- A. молоко/хлеб/колбаса
- B. молоко/хлеб/колбаса/яйца, яйца ещё! 
- C. молоко/хлеб/колбаса/ConcurrentModificationException
- D. ConcurrentModificationException

Что выведет следующий код?

```
List<String> list = new ArrayList<String>();  
list.add("молоко");  
list.add("хлеб");  
list.add("колбаса");  
list = list.subList(0, 2); //не надо колбасу!  
Stream<String> stream = list.stream();  
list.add("яйца, яйца ещё!");  
stream.forEach(System.out::println);
```

- A. молоко/хлеб/колбаса
- B. молоко/хлеб/колбаса/яйца, яйца ещё!
- C. молоко/хлеб/колбаса/ConcurrentModificationException
- D. молоко/хлеб/яйца, яйца ещё!

Что выведет следующий код?

```
List<String> list = new ArrayList<String>();  
list.add("молоко");  
list.add("хлеб");  
list.add("колбаса");  
list = list.subList(0, 2); //не надо колбасу!  
Stream<String> stream = list.stream();  
list.add("яйца, яйца ещё!");  
stream.forEach(System.out::println);
```

- A. молоко/хлеб/колбаса
- B. молоко/хлеб/колбаса/яйца, яйца ещё!
- C. молоко/хлеб/колбаса/ConcurrentModificationException
- D. молоко/хлеб/яйца, яйца ещё!


В чём разница между строчками 1 и 2?

```
public void killAll(){  
    ExecutorService ex = Executors.newSingleThreadExecutor();  
    List<String> sentence = Arrays.asList("Казнить");  
    ex.submit(() -> Files.write(Paths.get("Приговор.txt"), sentence) ); // 1  
    ex.submit(() -> { Files.write(Paths.get("Приговор.txt"), sentence); }); // 2  
}
```

- A. 1 компилируется, 2 нет
- B. 2 компилируется, 1 нет
- C. Что в лоб, что по лбу, обе нормально сработают.
- D. Без разницы, обе не компилируются.

В чём разница между строчками 1 и 2?

```
public void killAll(){  
    ExecutorService ex = Executors.newSingleThreadExecutor();  
    List<String> sentence = Arrays.asList("Казнить");  
    ex.submit(() -> Files.write(Paths.get("Приговор.txt"), sentence) ); // 1  
    ex.submit(() -> { Files.write(Paths.get("Приговор.txt"), sentence); }); // 2  
}
```

- A. 1 компилируется, 2 нет 
- B. 2 компилируется, 1 нет
- C. Что в лоб, что по лбу, обе нормально сработают.
- D. Без разницы, обе не компилируются.

Что произойдёт после выполнения?

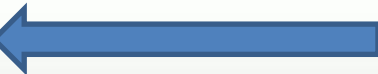
```
Map<String, String> oldSchool = initOldSchool();  
// oldSchool = {buildTool=maven, lang=java, IOC=jee}  
Map<String, String> proper = initКакНадо();  
// proper = {buildTool=gradle, lang=groovy, IOC=spring}  
  
oldSchool.replaceAll(proper::put);
```

- A. Мапы поменяются местами
- B. Обе станут олдскульными
- C. Обе станут как надо
- D. Поучитесь программировать! Это вообще не скомпилируется!



Что произойдёт после выполнения?

```
Map<String, String> oldSchool = initOldSchool();  
// oldSchool = {buildTool=maven, lang=java, IOC=jee}  
Map<String, String> proper = initКакНадо();  
// proper = {buildTool=gradle, lang=groovy, IOC=spring}  
  
oldSchool.replaceAll(proper::put);
```

- A. Мапы поменяются местами 
- B. Обе станут олдскульными
- C. Обе станут как надо
- D. Поучитесь программировать! Это вообще не скомпилируется!

Какой вариант сработает?

```
interface Кот{ default void мяукать() {System.out.println("мяу");}}  
interface Пёс{ default void лаять() {System.out.println("гав");}}  
  
public static void main(String[] args) {  
    class Котопёс implements Кот, Пёс {}  
    test(new Котопёс());  
}
```

```
static void test(Object obj) {  
    def x = (?)obj;  
    x.мяукать();  
    x.лаять();  
}
```

Какой вариант работает?

```
interface Кот{ default void мяукать() {System.out.println("мяу");}}  
interface Пёс{ default void лаять() {System.out.println("гав");}}
```

```
public static void main(String[] args) {  
    class Котопёс implements Кот, Пёс {}  
    test(new Котопёс());  
}
```

```
static void test(Object obj) {  
    // А. Это работает?  
    Кот & Пёс x = (Кот & Пёс) obj;  
    x.мяукать();  
    x.лаять();  
}
```

```
static void test(Object obj) {  
    // В. Это работает?  
    ((Consumer<? extends Кот & Пёс>)(x -> {  
        x.мяукать();  
        x.лаять();  
    })).accept((Кот & Пёс)obj);  
}
```

```
static void test(Object obj) {  
    // С. Это работает?  
    Optional.of((Кот & Пёс) obj)  
        .ifPresent(x -> {  
            x.мяукать();  
            x.лаять();  
        });  
}
```

// D. Да вы упоролись, в моей Джаве такого не бывает!



Что выведет?

```
public class Test {  
    String str;  
  
    void run() {  
        str = "привет";  
        Supplier<String> s1 = str::toUpperCase;  
        Supplier<String> s2 = () -> str.toUpperCase();  
        str = "hello";  
        System.out.println(s1.get());  
        System.out.println(s2.get());  
    }  
}
```

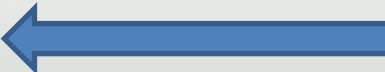
- A. ПРИВЕТ/ПРИВЕТ
- B. ПРИВЕТ/HELLO
- C. HELLO/ПРИВЕТ
- D. HELLO/HELLO



Что выведет?

```
public class Test {  
    String str;  
  
    void run() {  
        str = "привет";  
        Supplier<String> s1 = str::toUpperCase;  
        Supplier<String> s2 = () -> str.toUpperCase();  
        str = "hello";  
        System.out.println(s1.get());  
        System.out.println(s2.get());  
    }  
}
```

- A. ПРИВЕТ/ПРИВЕТ
- B. ПРИВЕТ/HELLO
- C. HELLO/ПРИВЕТ
- D. HELLO/HELLO





ЧТО ИЗ ЭТОГО НЕ КОМПИЛИРУЕТСЯ?

- A. **for** (; ;) { ; ; }
- B. **for** (; ;) ; ;
- C. { ; } **for** (; ;) { ; }
- D. ; **for** (; ;) ;
- E. Норм всё, чо!



ЧТО ИЗ ЭТОГО НЕ КОМПИЛИРУЕТСЯ?

A. `for (;;) {;;}`

B. `for (;;) ;;` 

C. `{;} for (;;) {;}`

D. `; for (;;) ;`

E. Норм всё, чо!