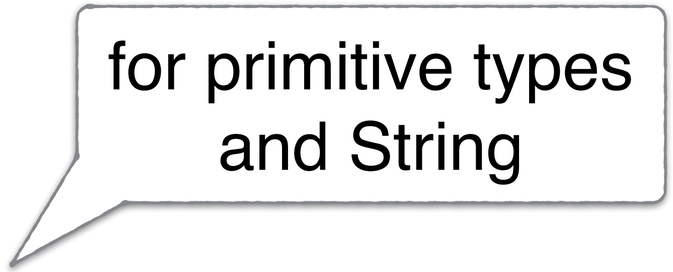
Constants

Constants

- `const` (for primitive types and String)
- `@JvmField` (eliminates accessors)

Compile-time constants




```
const val answer = 42
```

the value is inlined

@JvmField

exposes a Kotlin property as a field in Java

```kotlin
@JvmField
val property = MyClass()
```

```java
// the same as
// Java
public static final MyClass property = new MyClass();
```

@JvmField

exposes a Kotlin property as a field in Java

No getter!

```
@JvmField
val property = MyClass()
```

```
// the same as
// Java
public static final MyClass property = new MyClass();
```

`@JvmField` makes a property `static` if used at the top-level or inside object

```
class A {
    @JvmField
    val property = MyClass()
}
```

regular field generated

```
object B {
    @JvmField
    val property = MyClass()
}
```

`static` field generated

Which declaration(s) will expose `answer` as `static` field when used from Java?

```
object SuperComputer {
    val answer = 42
}
```

1.
```
@JvmStatic
val answer = 42
```

2.
```
@JvmField
val answer = 42
```

3.
```
const val answer = 42
```

Which declaration(s) will expose `answer` as static field when used from Java?

```
object SuperComputer {
    val answer = 42
}
```

1.
```
@JvmStatic
val answer = 42
```

2.
```
@JvmField
val answer = 42
```

3.
```
const val answer = 42
```

Property in an object

```
object SuperComputer {
    val answer = 42
}
```

```
// Java
SuperComputer.INSTANCE.getAnswer()
```

@JvmStatic

```
object SuperComputer {
    @JvmStatic
    val answer = 42
}
```

field isn't exposed

```
// Java
SuperComputer.getAnswer()
```

@JvmField

```
object SuperComputer {
    @JvmField
    val answer = 42
}


// Java
SuperComputer.answer
```

const

```
object SuperComputer {
    const val answer = 42
}
```

```
// Java
SuperComputer.answer
```

Which declaration(s) will inline the value of `answer` in the resulting bytecode?

```
object SuperComputer {
    val answer = 42
}
```

```
println(SuperComputer.answer)
```

```
System.out.println(42)
```

1.
```
@JvmStatic
val answer = 42
```

2.
```
@JvmField
val answer = 42
```

3.
```
const val answer = 42
```

Which declaration(s) will inline the value of `answer` in the resulting bytecode?

```
object SuperComputer {
    val answer = 42
}
```

```
println(SuperComputer.answer)
```

```
System.out.println(42);
```

1.
```
@JvmStatic
val answer = 42
```

2.
```
@JvmField
val answer = 42
```

3.
```
const val answer = 42
```

@JvmStatic

```
object SuperComputer {
    @JvmStatic
    val answer = 42
}

println(SuperComputer.answer)
```

```
System.out.println(SuperComputer.getAnswer());
```

@JvmField

```
object SuperComputer {
    @JvmField
    val answer = 42
}

println(SuperComputer.answer)
```

```
System.out.println(SuperComputer.answer);
```

const

```
object SuperComputer {
    const val answer = 42
}
```

```
println(SuperComputer.answer)
```

```
System.out.println(42);
```

Which declaration(s) will expose a top-level property as `static` field when used from Java?

```
val answer = 42
```

1.
```
val answer = 42
```

2.
```
@JvmField
val answer = 42
```

3.
```
const val answer = 42
```

Which declaration(s) will expose a top-level property as `static` field when used from Java?

```
val answer = 42
```

1.
```
val answer = 42
```

2.
```
@JvmField
val answer = 42
```

3.
```
const val answer = 42
```

Which declaration(s) will expose a top-level property as `static` field when used from Java?

```
val answer = 42
```

1.
```
val answer = 42
```
`Util.getAnswer()`

2.
```
@JvmField
val answer = 42
```
`Util.answer`

3.
```
const val answer = 42
```
`Util.answer`