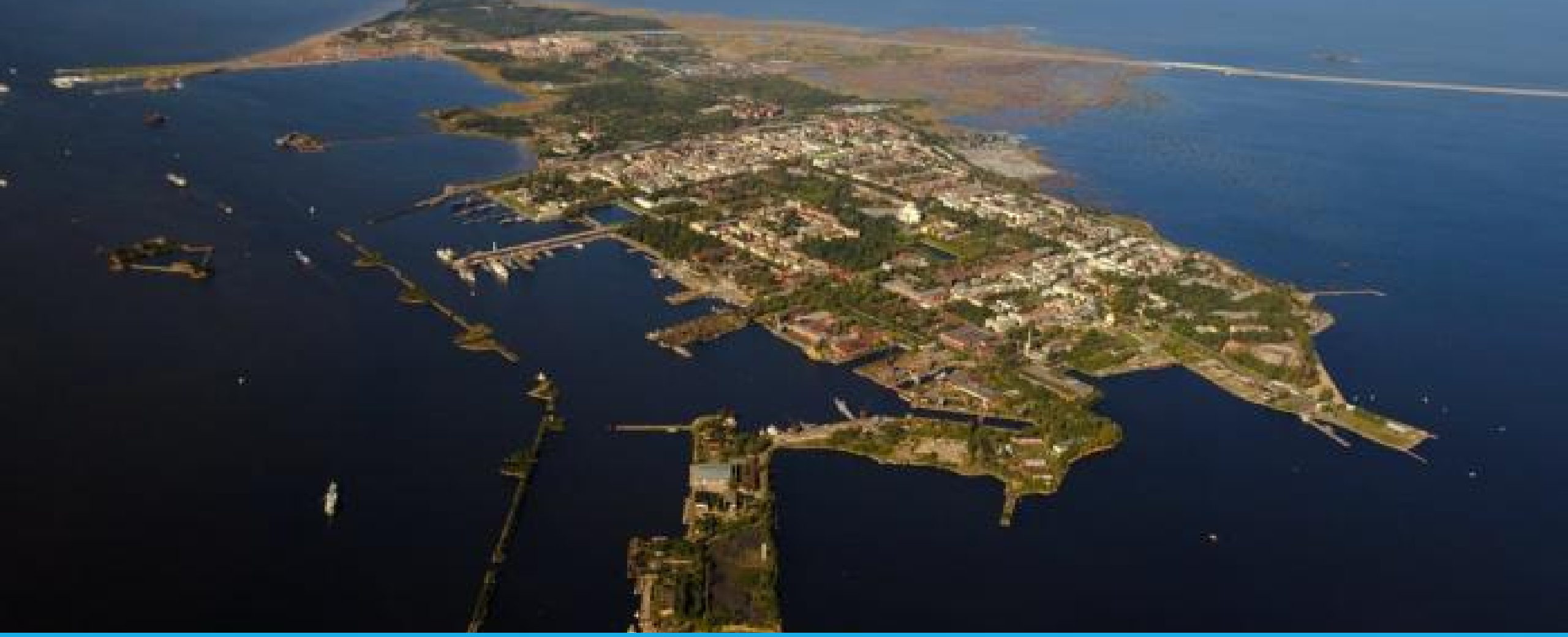
# Kotlin

WHAT JAVA MIGHT HAVE BEEN IF IT WERE WRITTEN TODAY



# What is it?

Island just west of Saint Petersburg, Russia in the Baltic Sea

# What is it?

- Programming language first revealed by JetBrains in 2011
- Statically-typed
- Runs on the JVM
- Fully interoperable with Java
- Version 1.0 released in February 2016
- Officially listed as a first-class language for Android development by Google in May 2017

# Open Source

JetBrains / kotlin

Watch 1,011 | Star 18,376 | Fork 1,955

Code | Pull requests 68 | Insights

The Kotlin Programming Language http://kotlinlang.org/

kotlin | programming-language | compiler | gradle-plugin | maven-plugin | kotlin-library | intellij-plugin

42,373 commits | 1,282 branches | 8,880 releases | 190 contributors

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

shiraji committed with semoro KT-20591: Show annotations in parameter info (#1358) ... Latest commit 11f9c05 2 days ago

| | | |
|---|---|---|
| .idea | Fix project scope "IDE" after migration to Gradle build | 5 days ago |
| android-studio | Prepare building plugins modules against 1.8 JDK | 2 years ago |
| annotations | Remove KotlinSignature annotations from project code | 2 years ago |
| ant | Fix tests in the new build infrastructure | a month ago |
| build-common | JS: fix copying functions with default arguments across interfaces | 10 days ago |
| buildSrc | Do not fail build on TC when some tests fail, just report problem | 5 days ago |
| compiler.tests-common | Rename android-annotations -> kotlin-annotations-android | 18 days ago |
| compiler | Minor, use extracted variable in LateinitIntrinsics.kt | 2 days ago |
| core | Don't resort deserialized descriptors based on renderer, preserve pro... | 2 days ago |
| custom-dependencies/protobuf-lite | Declare shared shadow plugin version and use it consistently | a month ago |

# Language Goals

- Be useful
  - Developed by developers for developers
  - Can be used for server-side, web, or Android applications. Native support is currently being worked on.
- Be safe
  - Support for non-nullable types
  - Can be introduced gradually to existing Java projects
- Be concise
  - Their estimates say that there should be about a 40% decrease in lines of code compared to Java.

# Hello, World!

```java
public static void main(String[] args) {
  System.out.println("Hello, World!");
}
```

```fun
fun main(args: Array<String>) {
    println("Hello, World!")
}
```

# Hello, World!

```
public static void main(String[] args) {
  System.out.println("Hello, " + args[0] + "!");
}
```

```
fun main(args: Array<String>) {
    println("Hello, ${args.first()}!")
}
```

# Convenient Immutability

```
public static void main(String[] args) {
  final String name = "Kyle";
  name = "Bob"; // Compilation Error

  String dayOfWeek = "Monday";
  dayOfWeek = "Tuesday";
}
```

# Convenient Immutability

```
fun main(args: Array<String>) {
  val name = "Kyle"
  name = "Bob" // Compilation Error

  var dayOfWeek = "Monday"
  dayOfWeek = "Tuesday"
}
```

# Convenient Immutability

```
public static void main(String[] args) {
  List<Integer> ages = Collections.unmodifiableList(
    Arrays.asList(22, 34, 49)
  );
  ages.add(25); // Compiles - Runtime Exception
}
```

# Convenient Immutability

```
fun main(args: Array<String>) {
  val ages = listOf(22, 34, 49)
  ages.add(25) // Compilation error

  var mutableAges = mutableListOf(22, 34, 49)
  mutableAges.add(25)
}
```

# Classes and Properties

```java
class Book {
  private String title;
  private String author;

  public Book(String title, String author) {
    this.title = title;
    this.author = author;
  }

  public String getTitle() {
    return title;
  }

  public String getAuthor() {
    return author;
  }
}
```

```java
Book aBook = new Book("The Hobbit",
  "J. R. R. Tolkien“);
System.out.println(aBook.getTitle());
```

# Classes and Properties

```
class Book(val title: String, val author: String)

fun main(args: Array<String>) {
  val aBook = Book("The Hobbit", "J. R. R. Tolkien")
  println(aBook.title)
}
```

# Data Classes

```
data class Book(val title: String, val author: String)

fun main(args: Array<String>) {
  val hobbit = Book("The Hobbit", "J. R. R. Tolkien")
  val lotr = hobbit.copy(title = "The Lord of the Rings")
  println(lotr.author) // J. R. R. Tolkien
}
```

# Constructor Overloading

```
class Book {
  private String title;
  private String author;

  public Book (String title) {
    this(title, "Unknown");
  }

  public Book(String title, String author) {
    this.title = title;
    this.author = author;
  }
}
```

# Constructor Overloading

```
data class Book(val title: String,
                val author: String = "Unknown",
                val pages: Int)

fun main(args: Array<String>) {
  val book = Book(title = "The Hobbit", pages = 320)
  println(book.author) // Unknown
}
```

# Null Safety

```
private String formatBookAndAuthor(Book book) {
  String name = "Unknown";
  if (book.getAuthor() != null) {
    name = book.getAuthor().getName();
  }

  return book.getTitle() + " written by " + name;
}
```

# Null Safety

```
class Book(val title : String, val author: Author?)
class Author(val name: String)

fun formatBookAndAuthor(book: Book) : String {
  // val name = book.author.name - won't compile
  val name = book.author?.name ?: "Unknown"
  return "${book.title} written by $name"
}
```

# Extension Methods

```
public static void main(String[] args) {
  DateUtils.dateAsJulian(LocalDate.now());
}
```

# Extension Methods

```
fun main(args: Array<String>) {
  println(LocalDateTime.now().asJulian())
}

fun LocalDateTime.asJulian() : Int {
  return this.year * 1000 + this.dayOfYear
}
```

# Extension Methods

```
public static void main(String[] args) {
  Bill bill = new Bill(BigDecimal.valueOf(1000));
  Bill anotherBill = new Bill(BigDecimal.valueOf(250));

  Bill totalBill = addBills(bill, anotherBill);
  System.out.println(totalBill.getAmount());
}

public static Bill addBills(Bill bill, Bill anotherBill) {
  BigDecimal total = bill.getAmount().add(anotherBill.getAmount());
  return new Bill(total);
}
```

# Extension Methods

```
fun main(args: Array<String>) {
  val bill = Bill(BigDecimal.valueOf(1000))
  val anotherBill = Bill(BigDecimal.valueOf(250))

  val totalBill = bill.plus(anotherBill)
  println(totalBill.amount)
}

fun Bill.plus(anotherBill: Bill): Bill {
  return Bill(this.amount.add(anotherBill.amount))
}
```

# Operator Overloading

```
fun main(args: Array<String>) {
  val bill = Bill(BigDecimal.valueOf(1000))
  val anotherBill = Bill(BigDecimal.valueOf(250))

  val totalBill = bill + anotherBill
  println(totalBill.amount)
}

operator fun Bill.plus(anotherBill: Bill): Bill {
  return Bill(this.amount.add(anotherBill.amount))
}
```

# Operator Overloading

```
public static void main(String[] args) {
  Bill bill = new Bill(BigDecimal.valueOf(1000));
  Bill anotherBill = new Bill(BigDecimal.valueOf(250));

  System.out.println(bill.compareTo(anotherBill) < 0);
}
```

# Operator Overloading

```
fun main(args: Array<String>) {
  val bill = Bill(BigDecimal.valueOf(1000))
  val anotherBill = Bill(BigDecimal.valueOf(250))

  println(bill < anotherBill)
}

operator fun Bill.compareTo(anotherBill : Bill) : Int {
  return this.amount.compareTo(anotherBill.amount)
}
```

# Type Casting

```
public static void main(String[] args) {
  Object obj = "UPPERCASE";
  if (obj instanceof String) {
    System.out.println(((String) obj).toLowerCase());
  }
}
```

# Type Casting

```
fun main(args: Array<String>) {
  val obj : Any = "UPPERCASE";
  if (obj is String) {
    println(obj.toLowerCase())
  }
}
```

# Java Interoperability

Call Kotlin class from Java

Call Java class from Kotlin

# Anything cool with Lambdas?