Algorithms

Produced by

Eamonn de Leastar (edeleastar@wit.ie)

Department of Computing, Maths & Physics
Waterford Institute of Technology

http://www.wit.ie

http://elearning.wit.ie

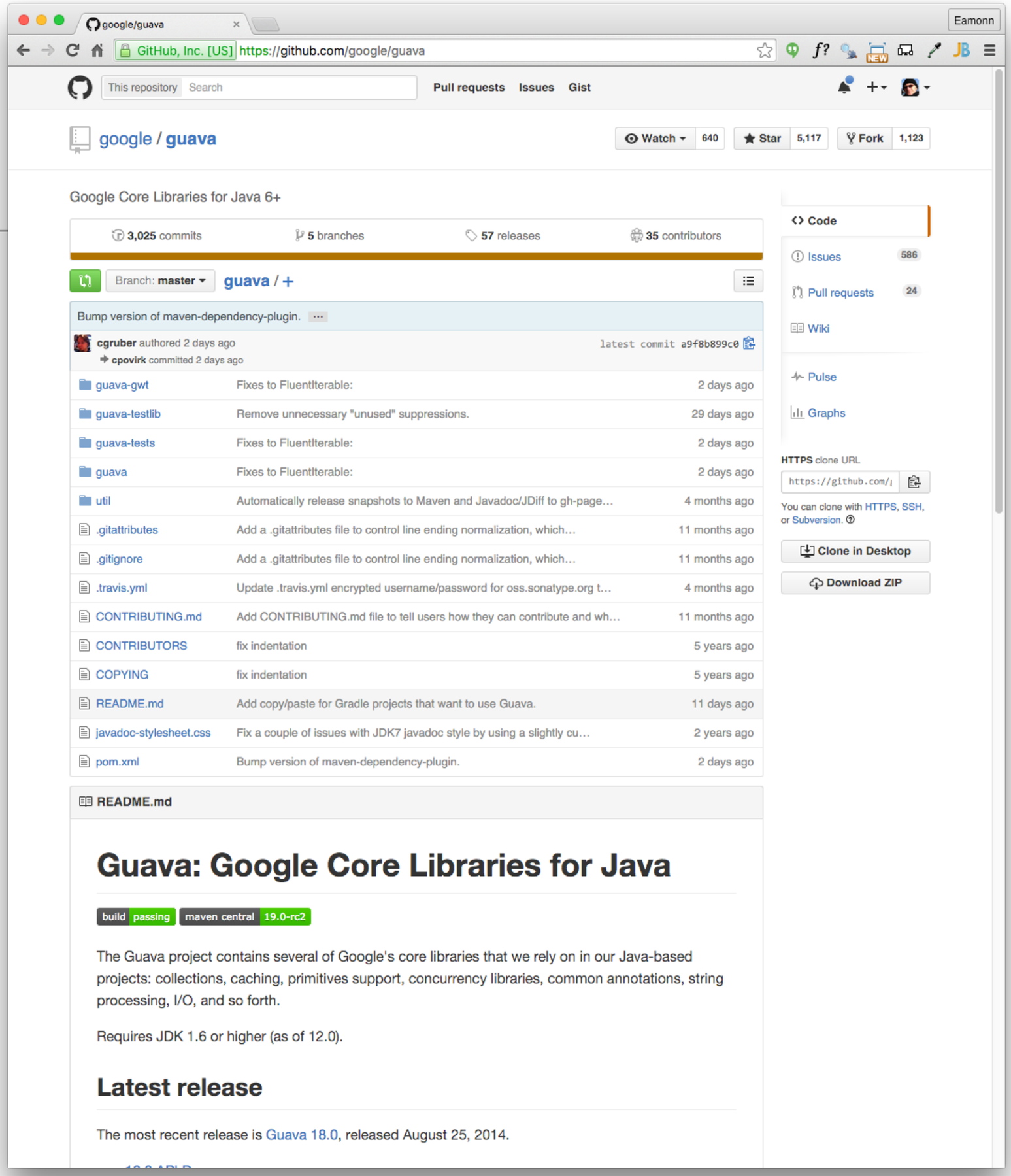
Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit

Pacemaker Lab02

Guava

- Google's Java Libraries
- Consider it an extension to the JDK to be included in all your projects



- Dominated but revisions and extensions the collections libraries

Home · google/guava Wiki

GitHub, Inc. [US] https://github.com/google/guava/wiki

This repository Search Pull requests Issues Gist

google / guava

Watch 640 Star 5,117 Fork 1,123

Home

Colin Decker edited this page on Jun 8 · 3 revisions

User Guide

The Guava project contains several of Google's core libraries that we rely on in our Java-based projects: collections, caching, primitives support, concurrency libraries, common annotations, string processing, I/O, and so forth. Each of these tools really do get used every day by Googlers, in production services.

But trawling through Javadoc isn't always the most effective way to learn how to make best use of a library. Here, we try to provide readable and pleasant explanations of some of the most popular and most powerful features of Guava.

This wiki is a work in progress, and parts of it may still be under construction.

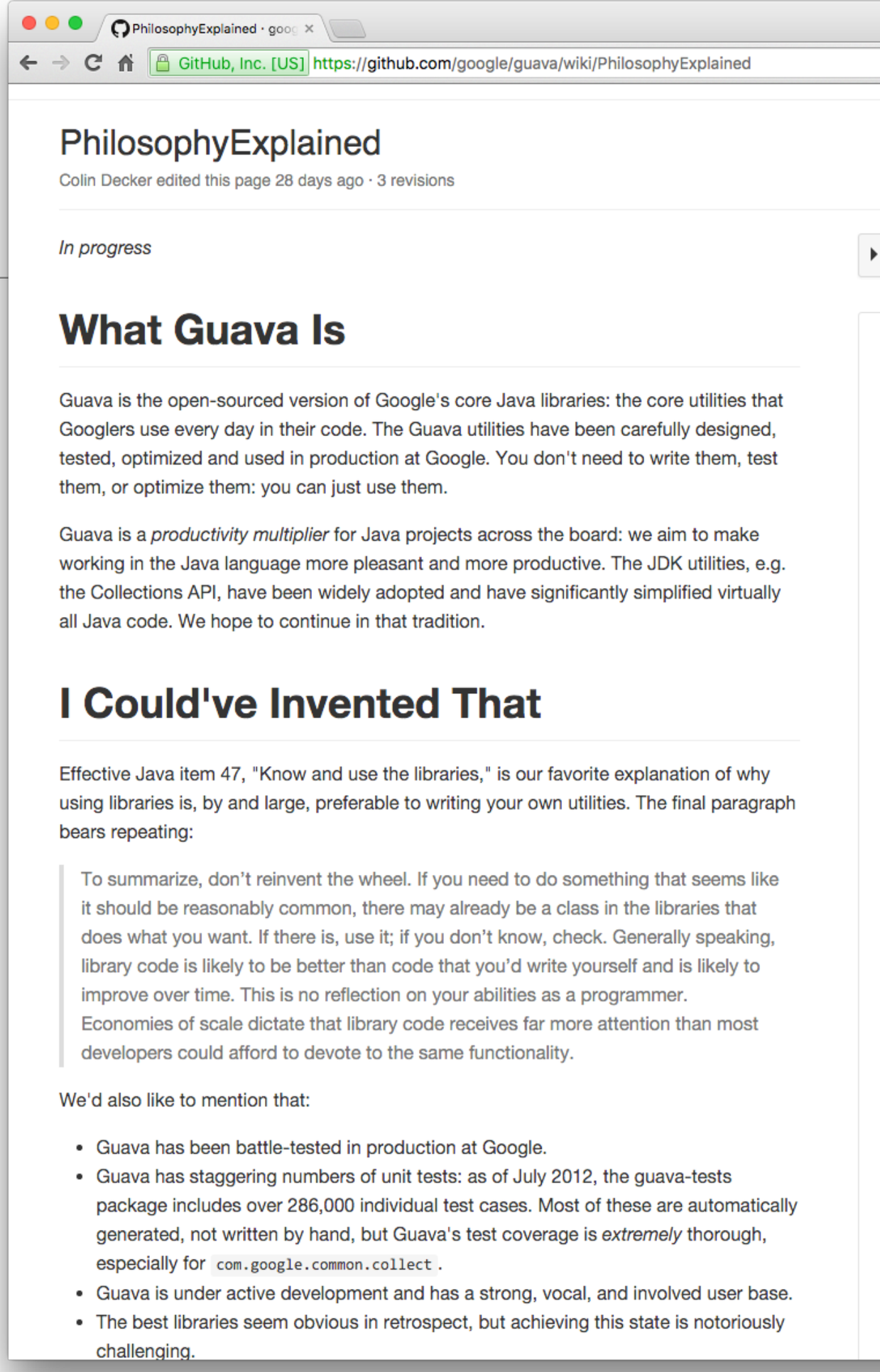
- Basic utilities: Make using the Java language more pleasant.
  - Using and avoiding null: `null` can be ambiguous, can cause confusing errors, and is sometimes just plain unpleasant. Many Guava utilities reject and fail fast on nulls, rather than accepting them blindly.
  - Preconditions: Test preconditions for your methods more easily.
  - Common object methods: Simplify implementing `Object` methods, like `hashCode()` and `toString()`.
  - Ordering: Guava's powerful "fluent `Comparator`" class.
  - Throwables: Simplify propagating and examining exceptions and errors.
- Collections: Guava's extensions to the JDK collections ecosystem. These are some of the most mature and popular parts of Guava.
  - Immutable collections, for defensive programming, constant collections, and improved efficiency.
  - New collection types, for use cases that the JDK collections don't address as well as they could: multisets, multimaps, tables, bidirectional maps, and more.
  - Powerful collection utilities, for common operations not provided in `java.util.Collections`.
  - Extension utilities: writing a `Collection` decorator? Implementing `Iterator`? We can make that easier.
- Caches: Local caching, done right, and supporting a wide variety of expiration behaviors.
- Functional idioms: Used sparingly, Guava's functional idioms can significantly

Pages 55

- Introduction
- Basic Utilities
  - Using/avoiding null
    - Optional
  - Preconditions
  - Ordering
    - Creation
    - Chaining
    - Application
  - Object methods
    - equals
    - hashCode
    - toString
    - compare/compareTo
  - Throwables
- Collections
  - Immutable collections
  - New collection types
    - Multiset
    - Multimap
    - BiMap
    - Table
    - ClassToInstanceMap
    - RangeSet
  - Utility Classes
    - Iterables
    - Lists
    - Sets
    - Maps
    - Multisets
    - Multimaps
    - Tables
  - Extension Utilities
    - Forwarding Decorators
    - PeekingIterator
    - AbstractIterator
- Caches
  - Applicability
  - Population

Guava Philosophy

"Guava is a **productivity multiplier** *for Java projects across the board: we aim to make working in the Java language more pleasant and more productive. The JDK utilities, e.g. the Collections API, have been widely adopted and have significantly simplified virtually all Java code. We hope to continue in that tradition."*



User

```java
package models;

import static com.google.common.base.MoreObjects.toStringHelper;
import com.google.common.base.Objects;

public class User
{
  static Long   counter = 0l;

  public Long   id;
  public String firstName;
  public String lastName;
  public String email;
  public String password;

  public User()
  {
  }

  public User(String firstName, String lastName, String email, String password)
  {
    this.id        = counter++;
    this.firstName = firstName;
    this.lastName = lastName;
    this.email = email;
    this.password = password;
  }

  public String toString()
  {
    return toStringHelper(this).addValue(firstName)
                               .addValue(lastName)
                               .addValue(password)
                               .addValue(email)
                               .toString();
  }

  @Override
  public int hashCode()
  {
     return Objects.hashCode(this.lastName, this.firstName, this.email, this.password);
  }
}
```

PacemakerAPI

- Store users indexed by email and id.

```
public class PacemakerAPI
{
  private Map<Long, User>     userIndex       = new HashMap<>();
  private Map<String, User>   emailIndex      = new HashMap<>();

  public Collection<User> getUsers ()
  {
    return userIndex.values();
  }

  public  void deleteUsers()
  {
    userIndex.clear();
    emailIndex.clear();
  }

  public User createUser(String firstName, String lastName, String email, String password)
  {
    User user = new User (firstName, lastName, email, password);
    userIndex.put(user.id, user);
    emailIndex.put(email, user);
    return user;
  }

  public User getUserByEmail(String email)
  {
    return emailIndex.get(email);
  }

  public User getUser(Long id)
  {
    return userIndex.get(id);
  }

  public void deleteUser(Long id)
  {
    User user = userIndex.remove(id);
    emailIndex.remove(user.email);
  }
}
```

Main

```
public class Main
{
  public static void main(String[] args) throws IOException
  {
    PacemakerAPI pacemakerAPI = new PacemakerAPI();

    pacemakerAPI.createUser("Bart", "Simpson",   "bart@simpson.com",  "secret");
    pacemakerAPI.createUser("Homer", "Simpson",  "homer@simpson.com", "secret");
    pacemakerAPI.createUser("Lisa", "Simpson",   "lisa@simpson.com", “ secret");

    Collection<User> users = pacemakerAPI.getUsers();
    System.out.println(users);

    User homer = pacemakerAPI.getUserByEmail("homer@simpson.com");
    System.out.println(homer);

    pacemakerAPI.deleteUser(homer.id);
    users = pacemakerAPI.getUsers();
    System.out.println(users);
  }
}
```