Can a Java Developer Learn Anything...

From Golang?

Grzegorz Piwowarek

@pivovarit

{ 4comprehension.com }

Vavr Lead | Java Champion | Oracle ACE

Independent Consultant/Trainer

distributed systems | microservices | async | reactive | java | idiomatic go

@pivovarit

”Imperative”

”Boilerplate-heavy”

”Clunky”

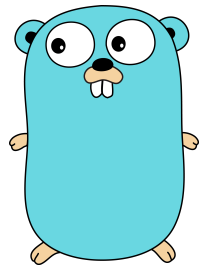
”Bloated”

...

Words people use to describe Java!

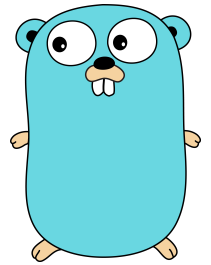
Golang is even more ”imperative”, ”boilerplate-heavy” than Java... and it’s proud of that!

So what...?



Golang's Philosophy

- Simplicity/Minimalism
- Explicitness
- Developer Productivity
- Pragmatism



Any advanced syntactic feature can be substituted with a finite number of for loops and if-else statements

```
for rowId := 0; rowId < len(pattern)-1; rowId++ {
  smudges := 0
  isRef := true
  for j := rowId; j >= 0; j-- {
    if rowId+rowId-j+1 < len(pattern) {
      diffs := rowDifferences(rowId+rowId-j+1, j, pattern)
      if diffs > 1 || (diffs > 0 && smudges == 1) {
        isRef = false
        break
      } else if diffs == 1 {
        smudges++
      }
    }
  }
  if !isRef {
    continue
  }
  if smudges == 1 {
    result := rowId + 1
    return true, result
  }
}
return false, -1
```

```go
func main() {
  urls := []string{"https://golang.org", "https://example.com"}

  results := make(chan string)
  start := time.Now()

  for _, url := range urls {
    go func(url string) {
      content, err := fetchURL(url)
      if err != nil {
        results <- fmt.Sprintf("Error fetching %s: %v", url, err)
        return
      }
      results <- fmt.Sprintf("Fetched %s: %s...", url, content)
    }(url)
  }

  for range urls {
    fmt.Println(<-results)
  }

  fmt.Printf("Execution time: %v\n", time.Since(start))
}
```

```go
package main

import (
  "fmt"
  "net/http"
)

func main() {
  http.HandleFunc("/hello", func(w http.ResponseWriter, r *http.R
    if r.Method == http.MethodGet {
      fmt.Fprintf(w, "Hello, World!")
    } else {
      http.Error(w, "Method not allowed", http.StatusMethodNotAll
    }
  })

  fmt.Println("Starting server on :8080...")
  err := http.ListenAndServe(":8080", nil)
  if err != nil {
    fmt.Printf("Error starting server: %v\n", err)
  }
}
```

Golang's Simplicity

- Three data structures: array/slice/maps
- One loop type
- No inheritance
- No exceptions
- No enums
- No mocking
- No advanced syntax
- No magical frameworks - wiring is explicit
- No ORMs by default

...but it does have nulls! (nils)

"It was the invention of the null reference in 1965...I call it my billion-dollar mistake."

https://www.reddit.com/r/golang/comments/pnjfg2/not_sure_which_one_describes_it_better/

So, what can we learn from Golang?

...

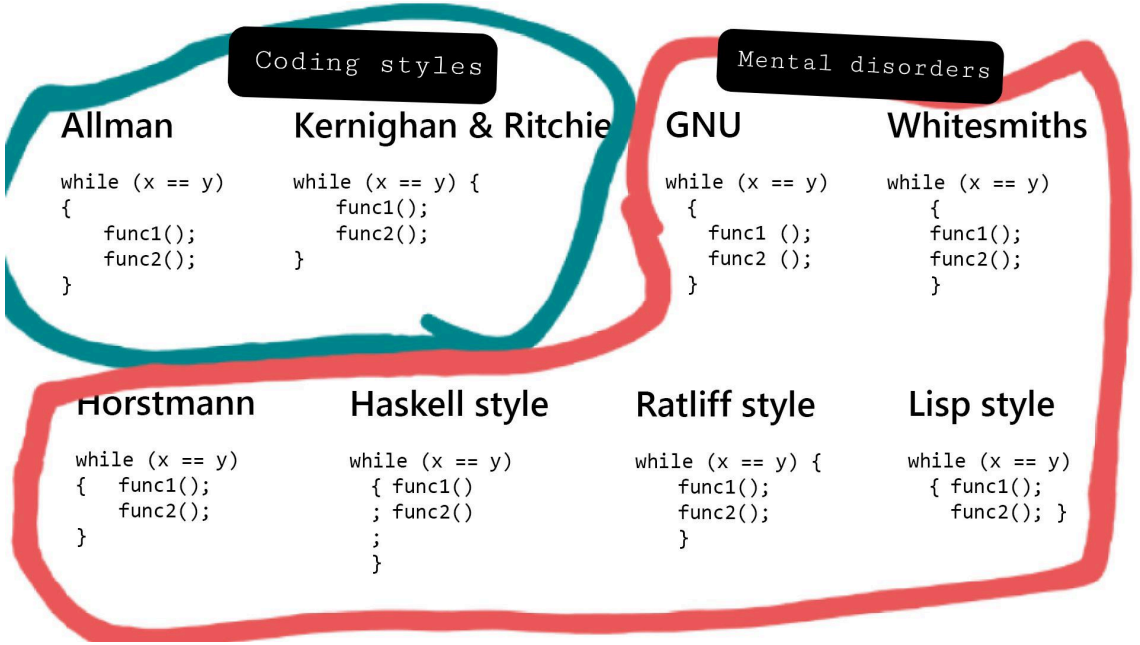
What's better? tabs or spaces?

"Gofmt formats Go programs. It uses tabs for indentation and blanks for alignment."

https://github.com/golang/go/blob/master/src/cmd/gofmt/doc.go

Less time debating. More time shipping.

Go Reduces Bikeshedding



https://www.reddit.com/r/ProgrammerHumor/comments/137drsn/i_cant_fathom_how_any_of_those_could_be_used/

No bikeshedding over:

code formatting (go fmt)

testing framework

build tools (go build)

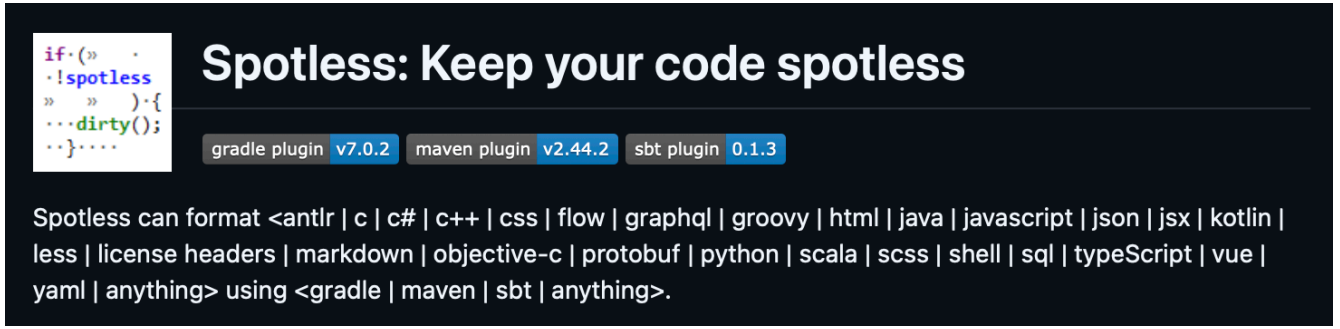
dependency management (Go Modules)

versioning scheme

linters (go vet)

...

Java Alternative: Spotless



https://github.com/diffplug/spotless

```xml
<plugin>
    <groupId>com.diffplug.spotless</groupId>
    <artifactId>spotless-maven-plugin</artifactId>
    <version>2.44.1</version>
    <configuration>
        <java>
            <googleJavaFormat/>
        </java>
    </configuration>
</plugin>
```

```
mvn spotless:check
```

Can be plugged into Maven/Gradle lifecycle

Unfortunately, it's configurable...

Available Code Styles:

- google-java-format
- eclipse jdt
- palantir-java-format
- formatAnnotations
- cleanthat

... now we can argue which code style to choose :)

Reproducible Builds

`go.sum`

built-in guarantee of **determinism and integrity**.

socials: @pivovarit

```
module github.com/example/project

go 1.23

require (
    github.com/labstack/echo/v4 v4.11.1
    golang.org/x/net v0.26.0
)
```

`go.mod` defines **what** you depend on.

```
github.com/labstack/echo/v4 v4.11.1 h1:Xf2m...
github.com/labstack/echo/v4 v4.11.1/go.mod h1:Dbz...
golang.org/x/net v0.26.0 h1:YeS...
golang.org/x/net v0.26.0/go.mod h1:Kf8...
```

`go.sum` records **exact cryptographic hashes** of every module and its *transitive dependencies*

- Ensures identical builds on every machine
- Detects tampered or re-published artifacts
- No extra tools or plugins required

“Maven verifies downloads.
Go verifies history.”

Reproducibility by default, not by configuration.

Strong Focus on Pragmatism

Example: Concurrency and Memory Models

```
 1 class Example {
 2
 3     private static boolean stop;
 4
 5     public static void main(String[] args)
 6       throws InterruptedException {
 7
 8         var thread = new Thread(() -> {
 9             int i = 0;
10             while (!stop) {
11                 i++;
12                 Integer.sum(i, 2); // ignore result
13             }
14         });
15
16     thread.start();
17     Thread.sleep(10); // play with this value
18
19     stop = true;
20     }
21 }
```

https://docs.oracle.com/javase/specs/jls/se8/html/jls-17.html#jls-17.4

```go
func main() {
    var ints = []int{0, 0}

    go func() { ints[0] = 1 }() // async
    go func() { ints[1] = 2 }() // async

    for !slices.Equal(ints, []int{1, 2}) {
        // busy spin ...
    }

    println("done, exiting!")
}
```

Is the main thread guaranteed to exit?

We need to consult Golang Memory Model

"Programs that modify data being simultaneously accessed by multiple goroutines must serialize such access."

"To serialize access, protect the data with channel operations or other synchronization primitives such as those in the sync and sync/atomic packages."

"(...) If you must read the rest of this document to understand the behavior of your program, you are being too clever. Don't be clever."

https://go.dev/ref/mem

Pragmatic, Not Clever Solution

```
func main() {
    var ints = []atomic.Int32{{}, {}}

    go func() { ints[0].Store(1) }() // async
    go func() { ints[1].Store(2) }() // async

    for ints[0].Load() != 1 && ints[1].Load() != 2 {
        // busy spin
    }

    println("done, exiting!")
}
```

```
 1 class Example {
 2
 3     private static boolean stop;
 4
 5     public static void main(String[] args)
 6       throws InterruptedException {
 7
 8         var thread = new Thread(() -> {
 9             int i = 0;
10             while (!stop) {
11                 i++;
12                 Integer.sum(i, 2); // ignore result
13             }
14         });
15         thread.start();
16         Thread.sleep(10); // play with this value
17
18         stop = true;
19     }
20 }
```

```
1  class Example {
2
3      private static final AtomicBoolean stop
4        = new AtomicBoolean();
5
6
7      public static void main(String[] args)
8        throws InterruptedException {
9
10         var thread = new Thread(() -> {
11             int i = 0;
12             while (!stop.get()) {
13                 i++;
14                 Integer.sum(i, 2); // ignore result
15             }
16         });
17         thread.start();
18         Thread.sleep(10); // play with this value
19
20         stop.set(true);
21     }
22 }
```

Java Memory Model - Nie Tylko Dla Orłów, Krzysztof Ślusarski

The Hidden Art of Thread-Safe Programming: Exploring java.util.concurrent, Heinz Kabutz

socials: @pivovarit

Pragmatic Concurrency Model

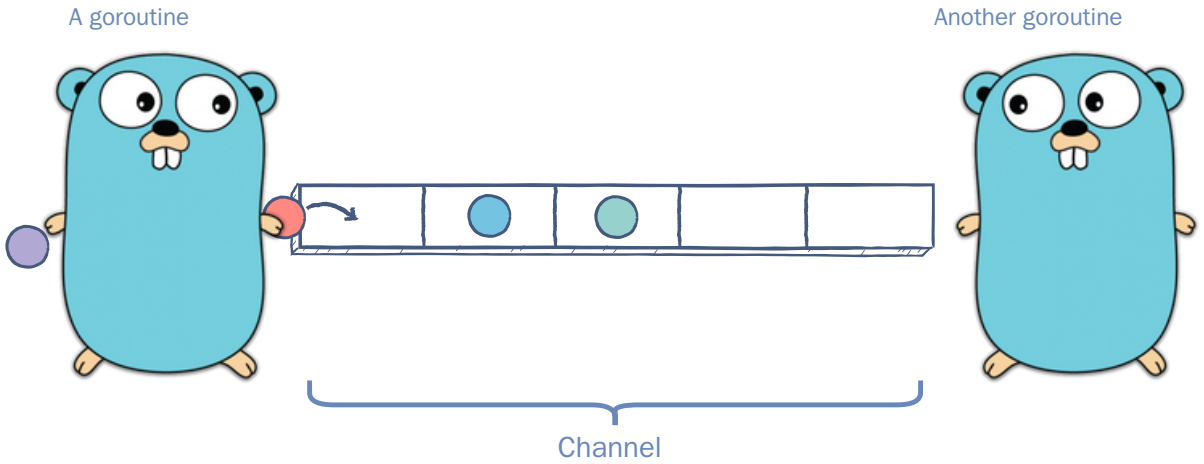
Goroutines and Channels

> *"Do not communicate by sharing memory; instead, share memory by communicating."*

Instead of shared state, goroutines communicate through channels

Effective Go

socials: @pivovarit



https://how.dev/answers/what-are-channels-in-golang

```go
func main() {
    ch := make(chan string)

    go func() {
        ch <- "ping"
    }()

    msg := <-ch
    fmt.Println(msg) // "ping"
}
```

when you hear "channel", think "special queue with dedicated syntax"

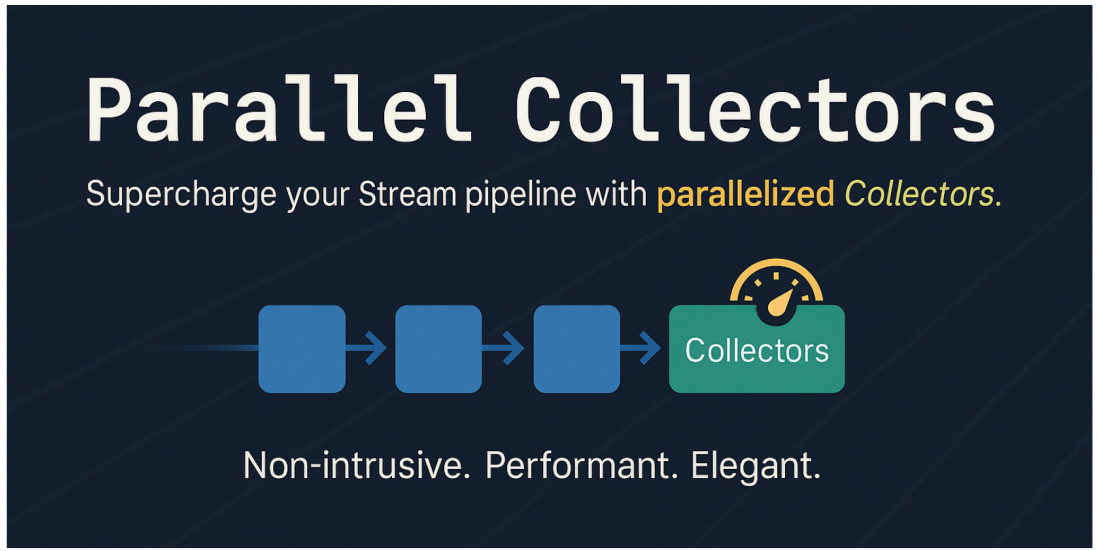
Benefits?

- Avoids low-level locking & race conditions
- Encourages isolation of state
- Scales naturally with concurrency
- Clearer communication patterns

Java Stream API Virtual-Threads-enabled Parallel Collectors

Overcoming limitations of standard Parallel Streams

build passing | pitest passing | maven-central v3.3.0



https://github.com/pivovarit/parallel-collectors

parallel-collectors use this trick internally

socials: @pivovarit

Dependency Inversion by Design

Golang's interfaces are not what you think they are

Go’s interfaces are designed with the consumer (caller) in mind rather than the producer

socials: @pivovarit

```
type MovieRentalService struct {
    rentalRepository RentalRepository
    movieDescriptionsRepository MovieDescriptionsRepository
    movieRepository MovieRepository
}
```

```
type MovieRentalService struct {
    rentalRepository interface {
        saveRentedMovie(userID, movieId string) error
        saveReturnedMovie(userID, movieId string) error
    }

    movieDescriptionsRepository interface {
        getMovieDescription(movieId string) (string, error)
    }

    movieRepository interface {
        getMovie(movieId string) (string, error)
    }
}
```

Interfaces are implicitly implemented (duck typing)

```
type PostgresMovieRepository struct {
  db *sql.DB
}

func (r PostgresMovieRepository) getMovie(id string) (string, err
    query := "SELECT title FROM movies WHERE id = $1"
  var title string
  err := r.db.QueryRow(query, id).Scan(&title)
  if err != nil {
    if err == sql.ErrNoRows {
      return "", fmt.Errorf("movie with id %s not found", id)
    }
    return "", err
  }
  return title, nil
}
```

```
type PostgresRentalRepository struct {
  db *sql.DB
}

func (r PostgresRentalRepository) saveRentedMovie(userID, movieId
  query := "INSERT INTO rentals (user_id, movie_id, rented_at) VA
  _, err := r.db.Exec(query, userID, movieId)
  return err
}

func (r PostgresRentalRepository) saveReturnedMovie(userID, movie
  query := "UPDATE rentals SET returned_at = NOW() WHERE user_id
  result, err := r.db.Exec(query, userID, movieId)
  if err != nil {
    return err
  }
  rowsAffected, _ := result.RowsAffected()
  if rowsAffected == 0 {
    return fmt.Errorf("no active rental found for user %s and mov
  }
  return nil
}
```

```go
type RESTMovieDescriptionsRepository struct {
  apiBaseURL string
  client     *http.Client
}

func (r RESTMovieDescriptionsRepository) getMovieDescription(movi
  url := fmt.Sprintf("%s/movies/%s/description", r.apiBaseURL, mo
  resp, err := r.client.Get(url)
  if err != nil {
    return "", err
  }
  defer resp.Body.Close()

  if resp.StatusCode != http.StatusOK {
    return "", fmt.Errorf("failed to fetch description: %s", resp
  }

  var result struct {
    Description string `json:"description"`
  }
  if err := json.NewDecoder(resp.Body).Decode(&result); err != ni
    return "", err
  }
  return result.Description, nil
}
```

```
service := MovieRentalService{
  rentalRepository:            PostgresRentalRepository{db},
  movieDescriptionsRepository: RESTMovieDescriptionsRepository{
      "https://api.example.com",
      &http.Client{}},
  movieRepository:             PostgresMovieRepository{db},
}
```

- Dependency Inversion
- Dependency Injection

Manual Dependency Injection

```
 1 rentalRepository := PostgresRentalRepository{db}
 2 descriptionsRepository := RESTMovieDescriptionsRepository{
 3     "https://api.example.com",
 4     &http.Client{}}
 5 movieRepository := PostgresMovieRepository{db}
 6
 7 movieRentalService := MovieRentalService{
 8     rentalRepository,
 9     descriptionsRepository,
10     movieRepository
11 }
```

Dependency Injection is about external supply of dependencies

Dependency Injection Containers are just one way to achieve this

But how to test it without mocks?

Disclaimer: Mocks vs Fakes

Fake → acts like the real thing (e.g. in-memory DB)

Mock → checks calls/interactions

If you call `Mockito.mock()` but never verify interactions, it's not a **Mock** - but a **Fake** (or even a **Dummy**).

Dependency Inversion Leverage

```
type InMemoryRentalRepository struct {
    // userID -> map[movieID]isReturned
    rentals map[string]map[string]bool
}

type InMemoryMovieRepository struct {
    // movieID -> title
    movies map[string]string
}

type InMemoryMovieDescriptionsRepository struct {
    // movieID -> description
    descriptions map[string]string
}
```

implementations omitted

We can do the same in Java!

```
class MovieRentalService {
    private final RentalHistoryRepository rentalHistoryRepository
    private final MovieRepository movieRepository;

    public void rentMovie(String login, UUID movieId) {
        // ...
        if (movieRepository.exists(movieId)) {
            rentalHistoryRepository.saveEvent(MovieRentalEventTyp
        }
    }

    public interface RentalHistoryRepository {
        void saveEvent(MovieRentalEventType eventType, String log
    }

    public interface MovieRepository {
        boolean exists(UUID movieId);
    }
}
```

Java requires explicit interface implementation, so this approach might generate dozens of boilerplate adapters

```
public interface RentalHistoryRepository {
    void saveEvent(MovieRentalEventType eventType, String login,
}
```

```
public interface MovieRepository {
    void save(Movie movie)
}
```

```
public enum MovieRentalEventType {RENTED, RETURNED}
```

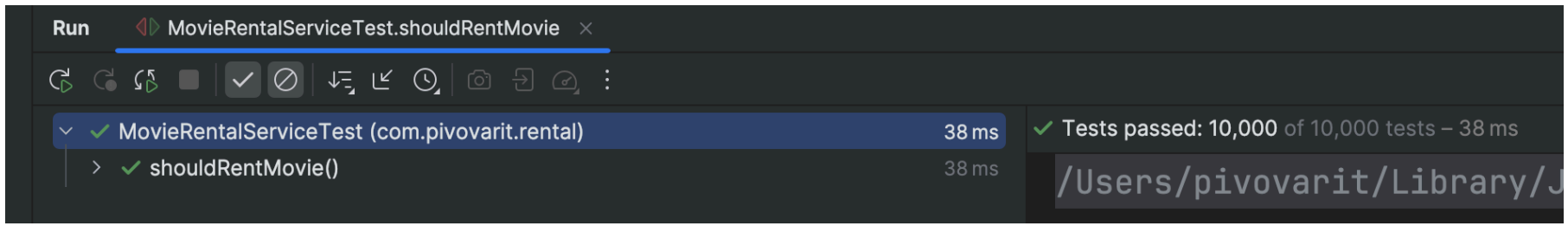
```
public static class InMemoryRentalHistoryRepository
  implements MovieRentalService.RentalHistoryRepository {

    private final List<MovieRentedEvent> events = Collections
      .synchronizedList(new ArrayList<>());

    @Override
    public void save(
      MovieRentalEventType eventType,
      String login, UUID id) {
        events.add(new MovieRentedEvent(login, id));
    }

    @Override
    public List<UUID> findRentedBy(String user) {
        return events.stream()
          .filter(e -> e.login().equals(user))
          .map(MovieRentedEvent::movieId)
          .toList();
    }

    record MovieRentedEvent(String login, UUID movieId) {
    }
}
```

```
@Test
void shouldRentMovie() {
    MovieRentalService service = new MovieRentalService(
      new InMemoryRentalHistoryRepository(),
      new InMemoryMovieRepository());

    var movieId = UUID.randomUUID();
    var user = "foo@bar.com";

    service.rentMovie(user, movieId);

    assertThat(service.getRentedMovies(user))
      .containsExactly(movieId);
}
```

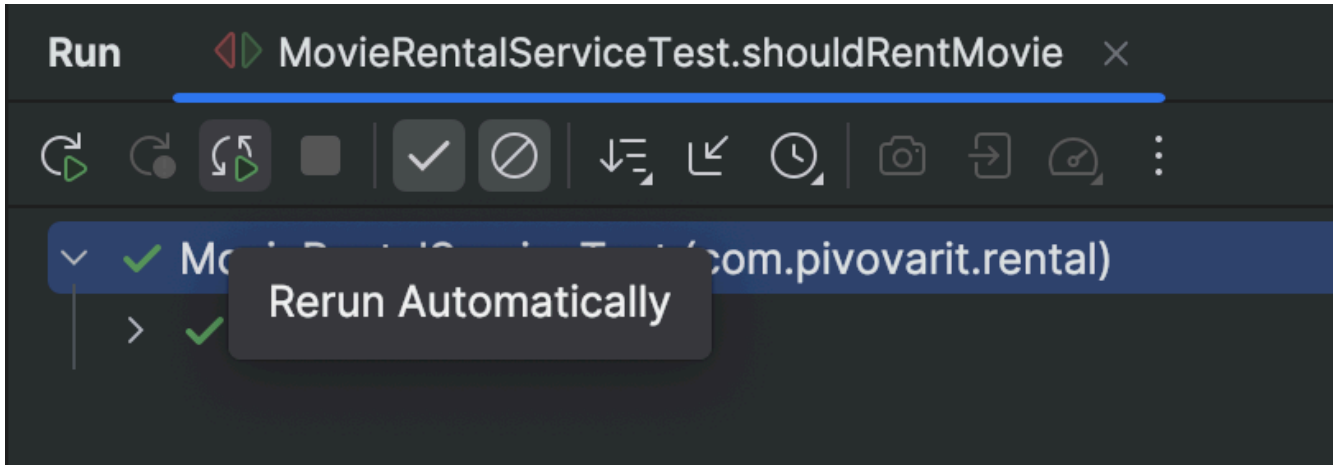
main advantage: in-memory tests are blazingly fast

```java
@RepeatedTest(10_000)
void shouldRentMovie() {
    MovieRentalService service = new MovieRentalService(
      new InMemoryRentalHistoryRepository(),
      new InMemoryMovieRepository());

    var movieId = UUID.randomUUID();
    var user = "foo@bar.com";

    service.rentMovie(user, movieId);

    assertThat(service.getRentedMovies(user))
      .containsExactly(movieId);
}
```

main advantage: in-memory tests are blazingly fast


Run
MovieRentalServiceTest.shouldRentMovie
MovieRentalServiceTest (com.pivovarit.rental)
38 ms
shouldRentMovie()
38 ms
Tests passed: 10,000 of 10,000 tests – 38 ms
/Users/pivovarit/Library/J

Near-compile-time Verification



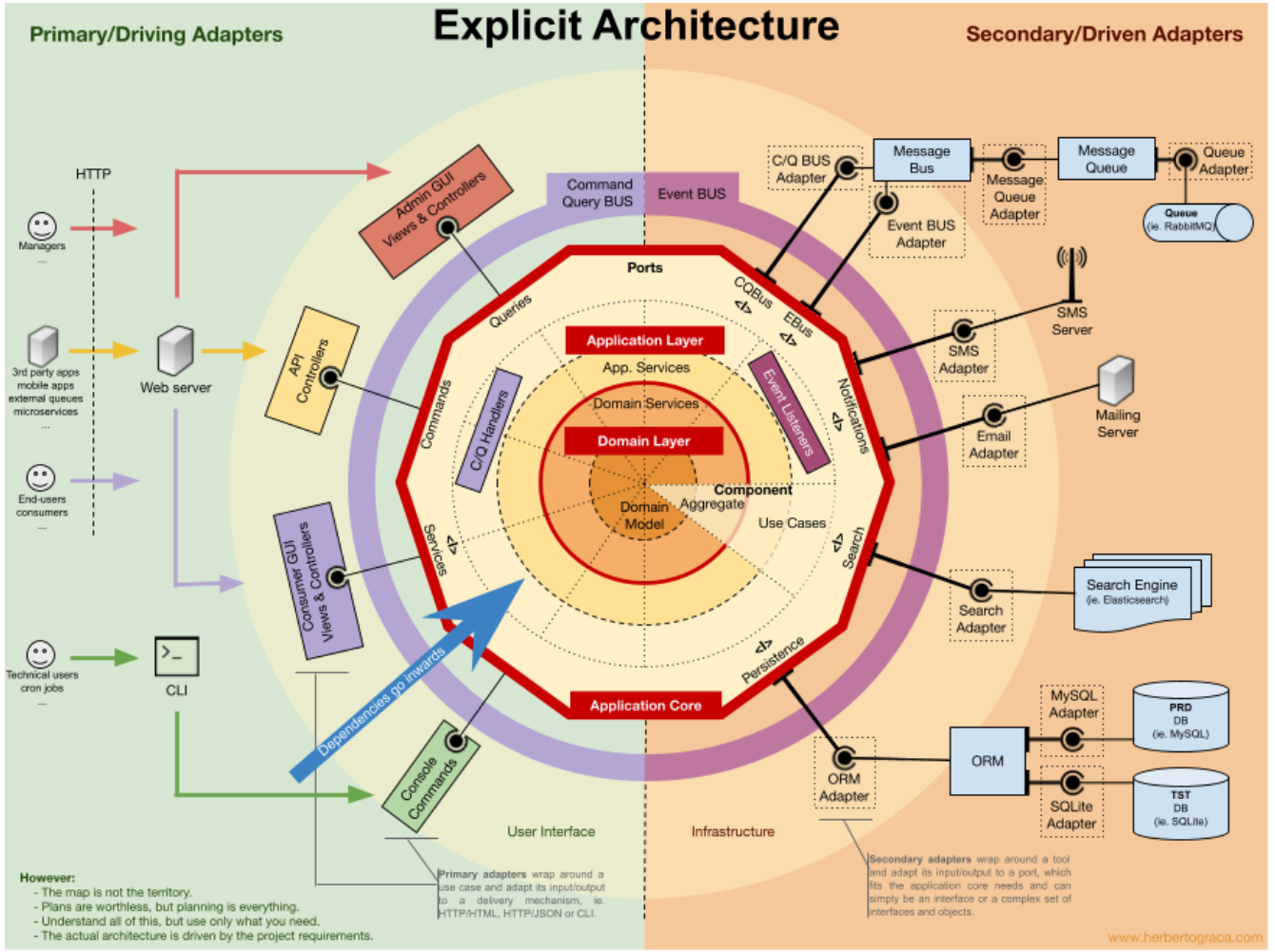
inspired by: compile-time sudoku solver in Scala

socials: @pivovarit

Works on every domain

```
public void rentMovie(String login, UUID movieId) {
    // ...
    if (movieRepository.exists(movieId)) {
//          rentalHistoryRepository.save(MovieRentalEventType.RENTED, login, movieId);
    } else {
        throw new IllegalArgumentException("movie does not exist");
    }
}

public List<UUID> getRentedMovies(String user) {
    return rentalHistoryRepository.findRentedBy(user);
}
```

This is Hexagonal Architecture (Ports and Adapters)



https://herbertograca.com/2017/11/16/explicit-architecture-01-ddd-hexagonal-onion-clean-cqrs-how-i-put-it-all-together/

socials: @pivovarit

Import/Dependency Cycles

Golang programs must be acyclic

socials: @pivovarit

```go
package main

import (
    "app/user"
)

func main() {
    user.Register("someone@example.com")
}
```

```go
package user

import (
    "app/email"
    "math/rand"
)

type User struct {
    ID    int
    Email string
}

func Register(email string) int {
    // ..
    NotifyUser(User{Email: email, ID: rand.Int()})
    return 42
}

func NotifyUser(u User) {
  email.SendWelcomeEmail(u)
}
```

```go
package email

import "app/user"

func Send(body, email string) {
  // ...
}

func SendWelcomeEmail(user user.User) {
  Send("Welcome!", user.Email)
}
```

```
> go build

package app
        imports app/user
        imports app/email
        imports app/user: import cycle not allowed
```

Acyclic Design

```
package email

func Send(body, email string) {
    // ...
}
```

```go
package user

import (
    "app/email"
    "math/rand"
)

type User struct {
    ID    int
    Email string
}

func Register(login string) int {
    // ..
    user := User{Email: login, ID: rand.Int()}
    email.Send("Welcome!", user.Email)
    return 42
}
```

https://github.com/golang/go/issues/30247

Identifying Cycles in Java

with ArchUnit

```
@ArchTest
static final ArchRule shouldBeFreeOfCycles = slices()
  .matching("com.pivovarit.(**)..")
  .should().beFreeOfCycles()
  .as("the library should be free of cycles")
  .because("cycles are bad");
```

https://www.archunit.org/

Errors as Values

"Values can be programmed, and since errors are values, errors can be programmed. ***Errors are not like exceptions. There's nothing special about them****, whereas an unhandled exception can crash your program."*

"Errors are values", Rob Pike

```go
func foo(s string) (string, error) { ... }
```

```go
response, err := foo("42")
if err != nil {
    // ...
}
```

https://x.com/kitarp29/status/1746515758589596139?lang=bg&mx=2

socials: @pivovarit

Limited Options for Java

- Use Records to return errors alongside results
- Use Either from Vavr

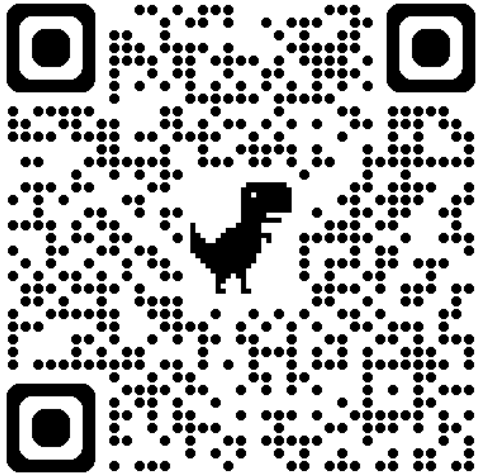
Using common sense is the ultimate Best Practice™.

Thank You!

Need help? Reach out! It's free.

@pivovarit

4comprehension.com



https://pivovarit.github.io/talks/learn-from-go

socials: @pivovarit