



# **WEEK4 - Feedbacks**

Mathias COUSTÉ  
25.02.2020

# 1

**teams did not compete...**

**(because of mvn fail)**

16  
*(out of 18)*



team finished the race

\o/



*Gottagofast*

**Week's champions**



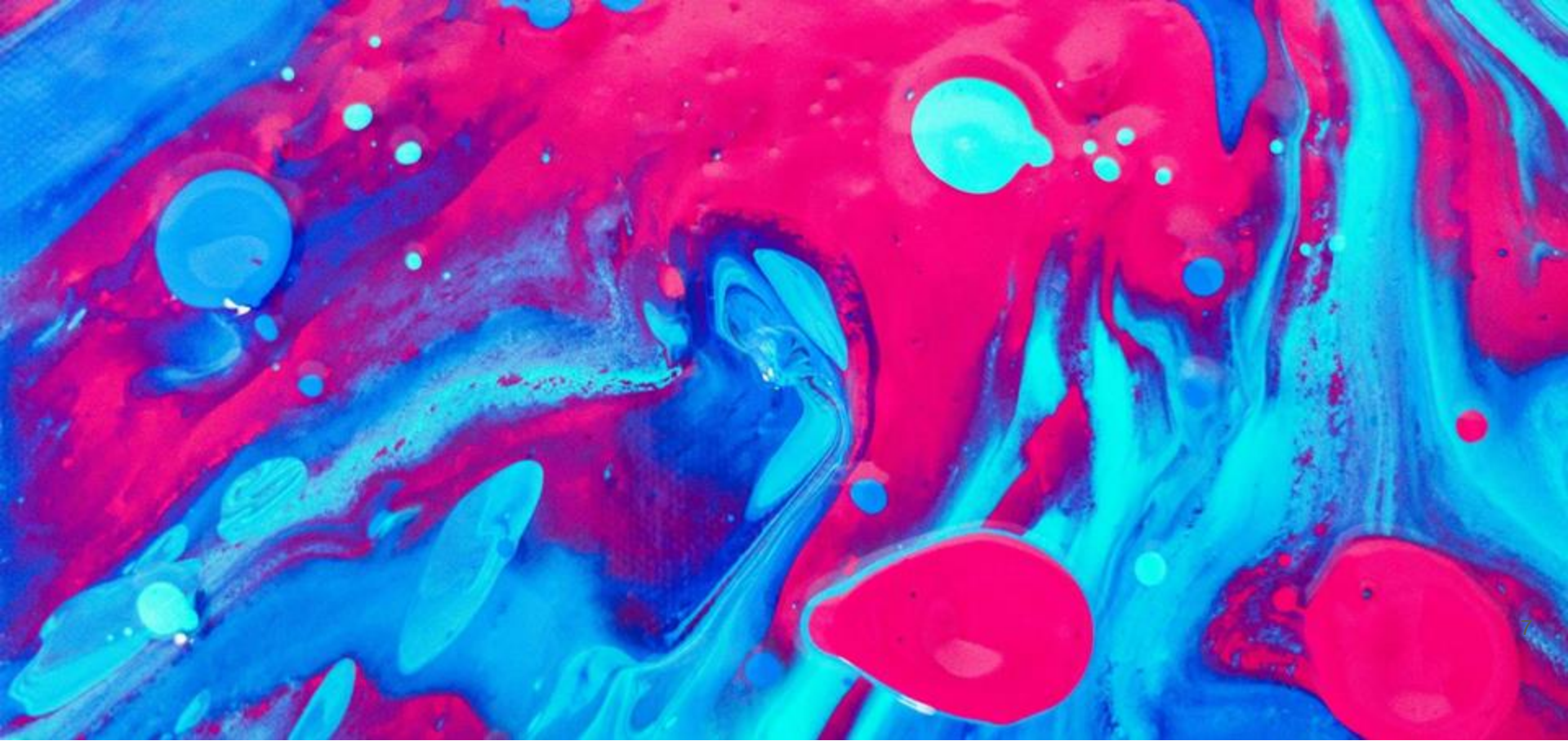
*Gottagofast*

Championship leaders

## NEXT WEEK

- Wind
- Sail
- LIFT\_SAIL action (*to open it*)
- LOWER\_SAIL action (*to close it*)

Deliver with tag **WEEK5** the *10th of March at 10PM.*



# **SOLID Principles**

Mathias COUSTÉ  
25.02.2020



**S** *ingle responsibility*

**O** *pen/Closed*

**L** *iskov substitution*

**I** *nterface segregation*

**D** *ependency inversion*




A large, white, serif capital letter 'S' is positioned on the left side of the image, set against a solid green background.

**Single responsibility**

# Single responsibility: definition


A class should have one and only one reason to change, meaning that a class should have only one job.

# Problem

```
▼  > Game
  ● run() : void
  ■ createTeamLogsFiles() : void
  ■ executeAction(PlayerAction) : void
  ■ initGame() : void
  ■ initGameForPlayer(Player) : void
  ■ activePlayers() : List<Player>
  ■ initRound() : List<PlayerAction>
  ■ nextRoundForPlayer(List<PlayerAction>) : Consumer<? super Player>
  ■ generateGameInitializationMessage(int, Player) : String
  ■ generateNextRoundGameMessage(Player) : String
  ■ visibleEntities(Sea, Ship) : List<SeaEntity>
  ■ isVisible(Ship, SeaEntity) : boolean
```

**How many responsibilities in this class?**

# Problem

▼  > Game

• `run() : void`

- `createTeamLogsFiles() : void`
- `executeAction(PlayerAction) : void`

- `initGame() : void`
- `initGameForPlayer(Player) : void`

■ `activePlayers() : List<Player>`

- `initRound() : List<PlayerAction>`
- `nextRoundForPlayer(List<PlayerAction>) : Consumer<? super Player>`

- `generateGameInitializationMessage(int, Player) : String`
- `generateNextRoundGameMessage(Player) : String`
- `visibleEntities(Sea, Ship) : List<SeaEntity>`
- `isVisible(Ship, SeaEntity) : boolean`

**How many responsibilities in this class?**

# Problem

▼  > Game

• `run() : void`

■ `createTeamLogsFiles() : void`

■ `executeAction(PlayerAction) : void`

■ `initGame() : void`

■ `initGameForPlayer(Player) : void`

■ `activePlayers() : List<Player>`

■ `initRound() : List<PlayerAction>`

■ `nextRoundForPlayer(List<PlayerAction>) : Consumer<? super Player>`

■ `generateGameInitializationMessage(int, Player) : String`


■ `generateNextRoundGameMessage(Player) : String`

■ `visibleEntities(Sea, Ship) : List<SeaEntity>`

■ `isVisible(Ship, SeaEntity) : boolean`

**How many responsibilities in this class?**

# Problem

▼  > Game

- `run() : void`
- `createTeamLogsFiles() : void`
- `executeAction(PlayerAction) : void`
- `initGame() : void`
- `initGameForPlayer(Player) : void`
- `activePlayers() : List<Player>`
- `initRound() : List<PlayerAction>`
- `nextRoundForPlayer(List<PlayerAction>) : Consumer<? super Player>`
- `generateGameInitializationMessage(int, Player) : String`
- `generateNextRoundGameMessage(Player) : String`
- `visibleEntities(Sea, Ship) : List<SeaEntity>`
- `isVisible(Ship, SeaEntity) : boolean`

**How many responsibilities in this class?**

# Problem

▼  > Game

• `run() : void`

■ `createTeamLogsFiles() : void`

■ `executeAction(PlayerAction) : void`

■ `initGame() : void`

■ `initGameForPlayer(Player) : void`

■ `activePlayers() : List<Player>`

■ `initRound() : List<PlayerAction>`

■ `nextRoundForPlayer(List<PlayerAction>) : Consumer<? super Player>`

■ `generateGameInitializationMessage(int, Player) : String`

■ `generateNextRoundGameMessage(Player) : String`

■ `visibleEntities(Sea, Ship) : List<SeaEntity>`

■ `isVisible(Ship, SeaEntity) : boolean`

**How many responsibilities in this class?**



# Resolution

Game

run() : void

createTeamLogsFiles() : void

executeAction(PlayerAction) : void

initGame() : void

initGameForPlayer(Player) : void

activePlayers() : List<Player>

initRound() : List<PlayerAction>

nextRoundForPlayer(List<PlayerAction>) : Consumer<? super Player>

generateGameInitializationMessage(int, Player) : String

generateNextRoundGameMessage(Player) : String

visibleEntities(Sea, Ship) : List<SeaEntity>

isVisible(Ship, SeaEntity) : boolean

Moved to Team class

**How many responsibilities in this class?**

# Resolution

Game

run() : void

createTeamLogsFiles() : void

executeAction(PlayerAction) : void

initGame() : void

initGameForPlayer(Player) : void

activePlayers() : List<Player>

initRound() : List<PlayerAction>

nextRoundForPlayer(List<PlayerAction>) : Consumer<? super Player>

generateGameInitializationMessage(int, Player) : String

generateNextRoundGameMessage(Player) : String

visibleEntities(Sea, Ship) : List<SeaEntity>

isVisible(Ship, SeaEntity) : boolean

Moved to Team class

New ActionExecutor class

**How many responsibilities in this class?**

# Resolution

Game

run() : void

createTeamLogsFiles() : void

executeAction(PlayerAction) : void

initGame() : void

initGameForPlayer(Player) : void

activePlayers() : List<Player>

initRound() : List<PlayerAction>

nextRoundForPlayer(List<PlayerAction>) : Consumer<? super Player>

generateGameInitializationMessage(int, Player) : String

generateNextRoundGameMessage(Player) : String

visibleEntities(Sea, Ship) : List<SeaEntity>

isVisible(Ship, SeaEntity) : boolean

Moved to Team class

New ActionExecutor class

Moved to helper class

**How many responsibilities in this class?**

# Resolution

Game

run() : void

createTeamLogsFiles() : void

executeAction(PlayerAction) : void

initGame() : void

initGameForPlayer(Player) : void

activePlayers() : List<Player>

initRound() : List<PlayerAction>

nextRoundForPlayer(List<PlayerAction>) : Consumer<? super Player>

generateGameInitializationMessage(int, Player) : String

generateNextRoundGameMessage(Player) : String

visibleEntities(Sea, Ship) : List<SeaEntity>

isVisible(Ship, SeaEntity) : boolean

Moved to Team class

New ActionExecutor class

Moved to helper class

New MessageBuilder class

**How many responsibilities in this class?**

# Resolution

Game

run() : void

createTeamLogsFiles() : void

executeAction(PlayerAction) : void

initGame() : void

initGameForPlayer(Player) : void

activePlayers() : List<Player>

initRound() : List<PlayerAction>

nextRoundForPlayer(List<PlayerAction>) : Consumer<? super Player>

generateGameInitializationMessage(int, Player) : String

generateNextRoundGameMessage(Player) : String

visibleEntities(Sea, Ship) : List<SeaEntity>

isVisible(Ship, SeaEntity) : boolean

Moved to Team class

New ActionExecutor class

Moved to helper class

New MessageBuilder class

Moved to Vision helper class

**How many responsibilities in this class?**

# 0

**Open / Closed**

# Open / Closed: definition

Objects or entities should be open for extension, but closed for modification.



# Problem

```
switch (action.getType()) {
case OAR:
    Optional<Oar> optOar = player.getShip().findEntityAt(sailor.getX(), sailor.getY(), Oar.class);
    if (optOar.isEmpty() || optOar.get().isUsed()) {
        logger.warn("Cannot execute action, no available oar in the sailor position");
        return;
    }
    break;
case MOVING:
    Moving moving = (Moving) action;
    if (!sailor.canMove(moving.getXDistance(), moving.getYDistance()) || !player.getShip().getDeck()
        .isIn(sailor.getX() + moving.getXDistance(), sailor.getY() + moving.getYDistance())) {
        logger.warn("Cannot execute action, no available oar in the sailor position");
        return;
    }
    break;
case TURN:
    Turn turn = (Turn) action;

    if (turn.getRotation() < -Math.PI / 4 || turn.getRotation() > Math.PI / 4) {
        logger.warn("Cannot execute action, turn rotation is out of range");
        return;
    }

    Optional<Rudder> optRudder = player.getShip().findEntityAt(sailor.getX(), sailor.getY(), Rudder.class);
    if (!optRudder.isPresent() || optRudder.get().isUsed()) {
        logger.warn("Cannot execute action, no available rudder in the sailor position");
        return;
    }
    break;
case LIFT_SAIL:
    Optional<Sail> optLiftSail = player.getShip().findEntityAt(sailor.getX(), sailor.getY(), Sail.class);
    if (!optLiftSail.isPresent() || optLiftSail.get().isUsed()) {
        logger.warn("Cannot execute action, no available sail in the sailor position");
        return;
    }
    break;
```

# Problem

```
switch (action.getType()) {
case OAR:
    Optional<Oar> optOar = player.getShip().findEntityAt(sailor.getX(), sailor.getY(), Oar.class);
    if (optOar.isEmpty() || optOar.get().isUsed()) {
        logger.warn("Cannot execute action, no available oar in the sailor position");
        return;
    }
    break;
case MOVING:
    Moving moving = (Moving) action;
    if (!sailor.canMove(moving.getXDistance(), moving.getYDistance()) || !player.getShip().getDeck()
        .isIn(sailor.getX() + moving.getXDistance(), sailor.getY() + moving.getYDistance())) {
        logger.warn("Cannot execute action, no available oar in the sailor position");
        return;
    }
    break;
case TURN:
    Turn turn = (Turn) action;

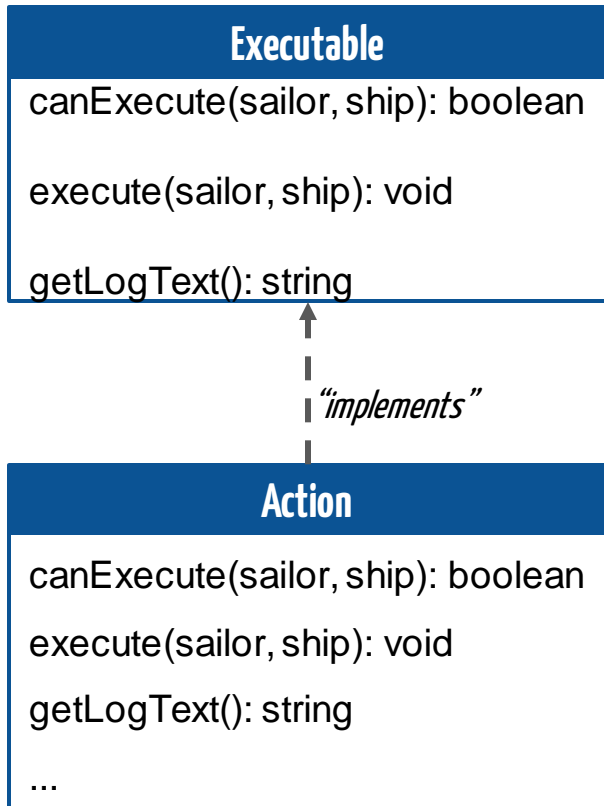
    if (turn.getRotation() < -Math.PI / 4 || turn.getRotation() > Math.PI / 4) {
        logger.warn("Cannot execute action, turn rotation is out of range");
        return;
    }

    Optional<Rudder> optRudder = player.getShip().findEntityAt(sailor.getX(), sailor.getY(), Rudder.class);
    if (!optRudder.isPresent() || optRudder.get().isUsed()) {
        logger.warn("Cannot execute action, no available rudder in the sailor position");
        return;
    }
    break;
case LIFT_SAIL:
    Optional<Sail> optLiftSail = player.getShip().findEntityAt(sailor.getX(), sailor.getY(), Sail.class);
    if (!optLiftSail.isPresent() || optLiftSail.get().isUsed()) {
        logger.warn("Cannot execute action, no available sail in the sailor position");
        return;
    }
    break;
```

This code needs to be updated every time I want to add a new Action.

With half of the actions implemented, this function is already 50 lines long :(

# Resolution



```
if (action.canExecute(sailor, player.getShip())) {
    action.execute(sailor, player.getShip());
} else {
    logger.warn("Cannot execute action" +
action.getLogText());
}
```



# L

# Liskov substitution

# Liskov substitution: definition

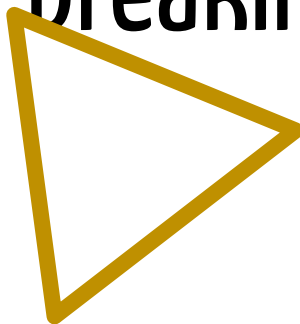
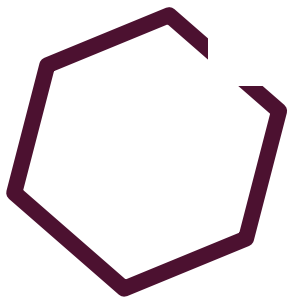
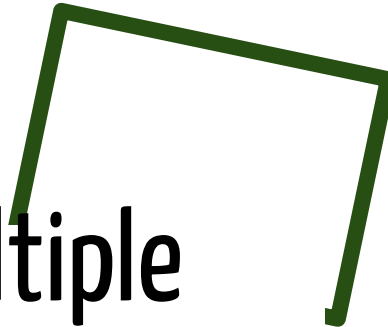
Let  $q(x)$  be a property provable about objects of  $x$  of type  $T$ . Then  $q(y)$  should be provable for objects  $y$  of type  $S$  where  $S$  is a subtype of  $T$ .

# Liskov substitution: definition

If you replace a class by one of its siblings or children,  
the program should keep working.

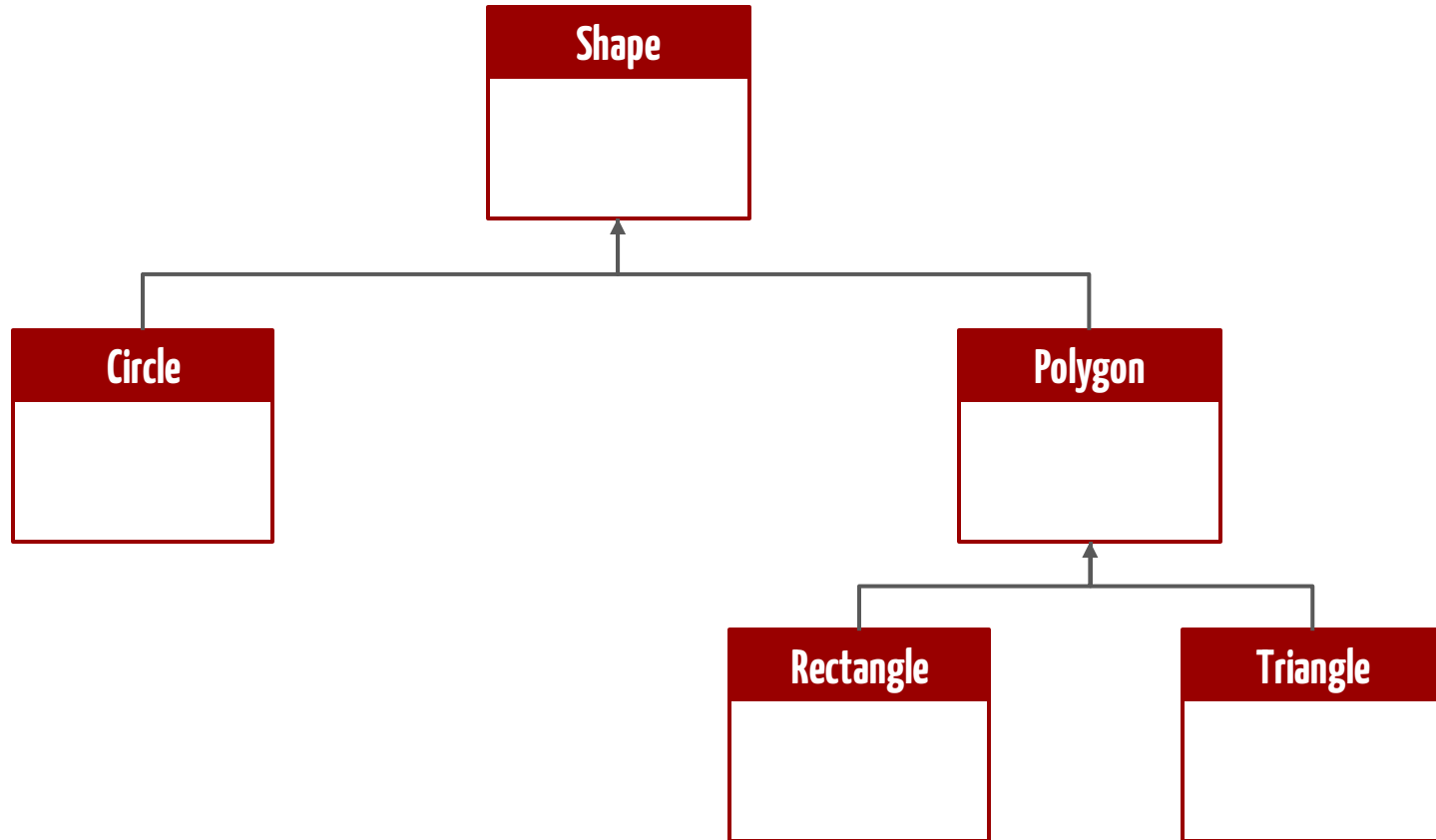
# Benefits

How to support multiple  
shapes in my game without  
breaking it?





# Benefits





# Interface segregation

# Interface segregation: definition

A client should never be forced to implement an interface that it doesn't use or clients shouldn't be forced to depend on methods they do not use.

# Problem

```
public interface ISeaEntity {  
    public Position getPosition();  
  
    public double distance(Positionable other);  
  
    public double getX();  
  
    public void setX(double x);  
  
    public double getY();  
  
    public void setY(double y);  
  
    public double getOrientation();  
  
    public void setOrientation(double orientation);  
  
    public double getNextX();  
  
    public void setNextX(double x);  
  
    public double getNextY();  
  
    public void setNextY(double y);  
  
    public double getNextOrientation();  
  
    public void setNextOrientation(double orientation);  
  
    public Shape getShape();  
}
```

***This interface matches all the requirements for sea entities...***

# Problem

```
public interface ISeaEntity {  
    public Position getPosition();  
  
    public double distance(Positionable other);  
  
    public double getX();  
  
    public void setX(double x);  
  
    public double getY();  
  
    public void setY(double y);  
  
    public double getOrientation();  
  
    public void setOrientation(double orientation);  
  
    public double getNextX();  
  
    public void setNextX(double x);  
  
    public double getNextY();  
  
    public void setNextY(double y);  
  
    public double getNextOrientation();  
  
    public void setNextOrientation(double orientation);  
  
    public Shape getShape();  
}
```

Can a reef move?

# Problem

```
public interface ISeaEntity {  
    public Position getPosition();  
  
    public double distance(Positionable other);  
  
    public double getX();  
  
    public void setX(double x);  
  
    public double getY();  
  
    public void setY(double y);  
  
    public double getOrientation();  
  
    public void setOrientation(double orientation);  
  
    public double getNextX();  
  
    public void setNextX(double x);  
  
    public double getNextY();  
  
    public void setNextY(double y);  
  
    public double getNextOrientation();  
  
    public void setNextOrientation(double orientation);  
  
    public Shape getShape();  
}
```

Does a sea flag need a shape?

# Resolution

```
public interface ISeaEntity {  
    public Position getPosition();  
  
    public double distance(Positionable other);  
  
    public double getX();  
  
    public void setX(double x);  
  
    public double getY();  
  
    public void setY(double y);  
  
    public double getOrientation();  
  
    public void setOrientation(double orientation);  
  
    public double getNextX();  
  
    public void setNextX(double x);  
  
    public double getNextY();  
  
    public void setNextY(double y);  
  
    public double getNextOrientation();  
  
    public void setNextOrientation(double orientation);  
  
    public Shape getShape();  
}
```

→ Positionnable



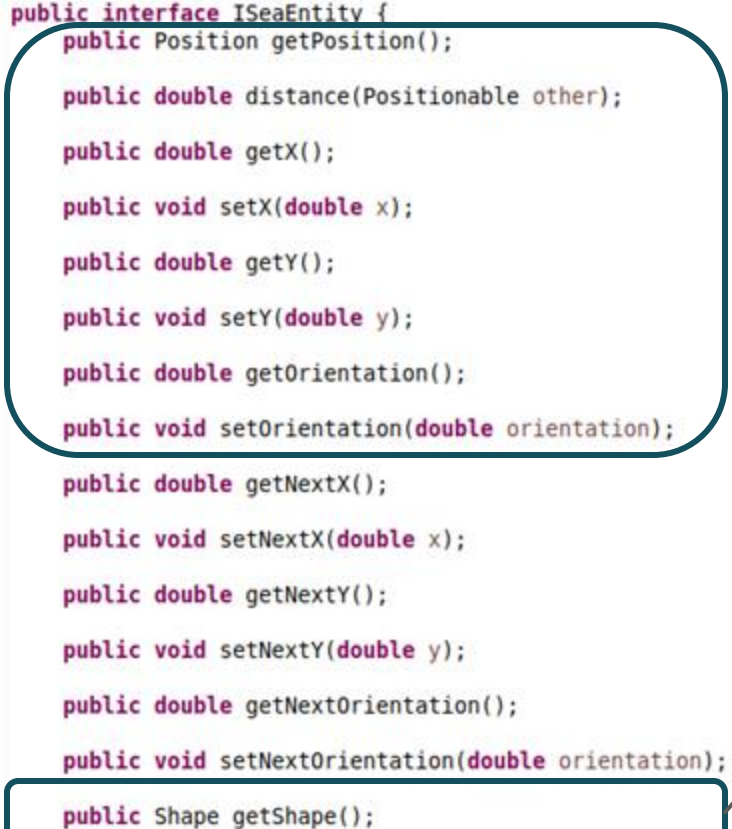
# Resolution

```
public interface ISeaEntity {  
    public Position getPosition();  
  
    public double distance(Positionable other);  
  
    public double getX();  
  
    public void setX(double x);  
  
    public double getY();  
  
    public void setY(double y);  
  
    public double getOrientation();  
  
    public void setOrientation(double orientation);  
  
    public double getNextX();  
    public void setNextX(double x);  
    public double getNextY();  
    public void setNextY(double y);  
    public double getNextOrientation();  
    public void setNextOrientation(double orientation);  
    public Shape getShape();  
}
```

→ Movable

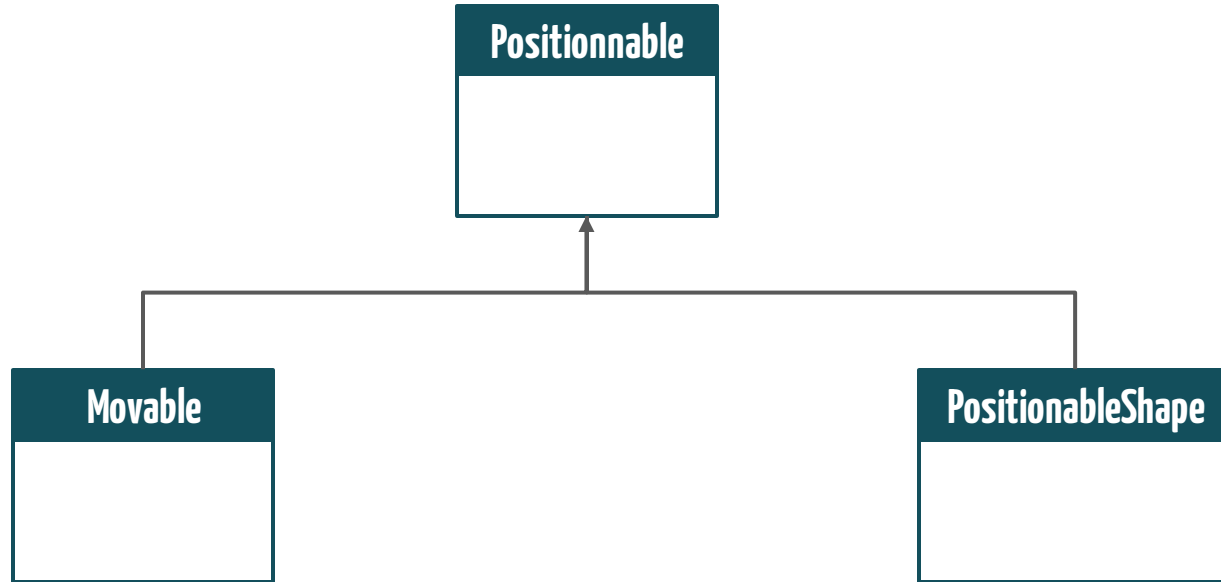
# Resolution

```
public interface ISeaEntity {  
    public Position getPosition();  
  
    public double distance(Positionable other);  
  
    public double getX();  
  
    public void setX(double x);  
  
    public double getY();  
  
    public void setY(double y);  
  
    public double getOrientation();  
  
    public void setOrientation(double orientation);  
  
    public double getNextX();  
  
    public void setNextX(double x);  
  
    public double getNextY();  
  
    public void setNextY(double y);  
  
    public double getNextOrientation();  
  
    public void setNextOrientation(double orientation);  
  
    public Shape getShape();  
}
```



PositionnableShape

# Resolution



# D

**Dependency inversion**

# Dependency inversion: definition

Entities must depend on abstractions not on concretions. It states that the high level module must not depend on the low level module, but they should depend on abstractions.

# Problem

```
public class Game {  
  
    private static Logger logger = Logger.getLogger(Game.class);  
  
    private RegattaResolver resolver;  
    private RegattaGameGoal goal;  
  
    private RoundMovementsRunner roundMovementsRunner;  
    private Sea sea;  
    private List<Player> players;  
  
    private int maxRounds;  
    private int currentRound;  
}
```

# Problem

```
public class Game {  
  
    private static Logger logger = Logger.getLogger(Game.class);  
  
    private RegattaResolver resolver;  
    private RegattaGameGoal goal;  
  
    private RoundMovementsRunner roundMovementsRunner;  
    private Sea sea;  
    private List<Player> players;  
  
    private int maxRounds;  
    private int currentRound;
```

# Problem

```
public class Game {  
  
    private static Logger logger = Logger.getLogger(Game.class);  
  
    private RegattaResolver resolver;  
    private RegattaGameGoal goal;  
  
    private RoundMovementsRunner roundMovementsRunner;  
    private Sea sea;  
    private List<Player> players;  
  
    private int maxRounds;  
    private int currentRound;  
}
```

Do I need to create a *BattleGame* class when I will add the *BattleGameGoal* and *BattleResolver*?

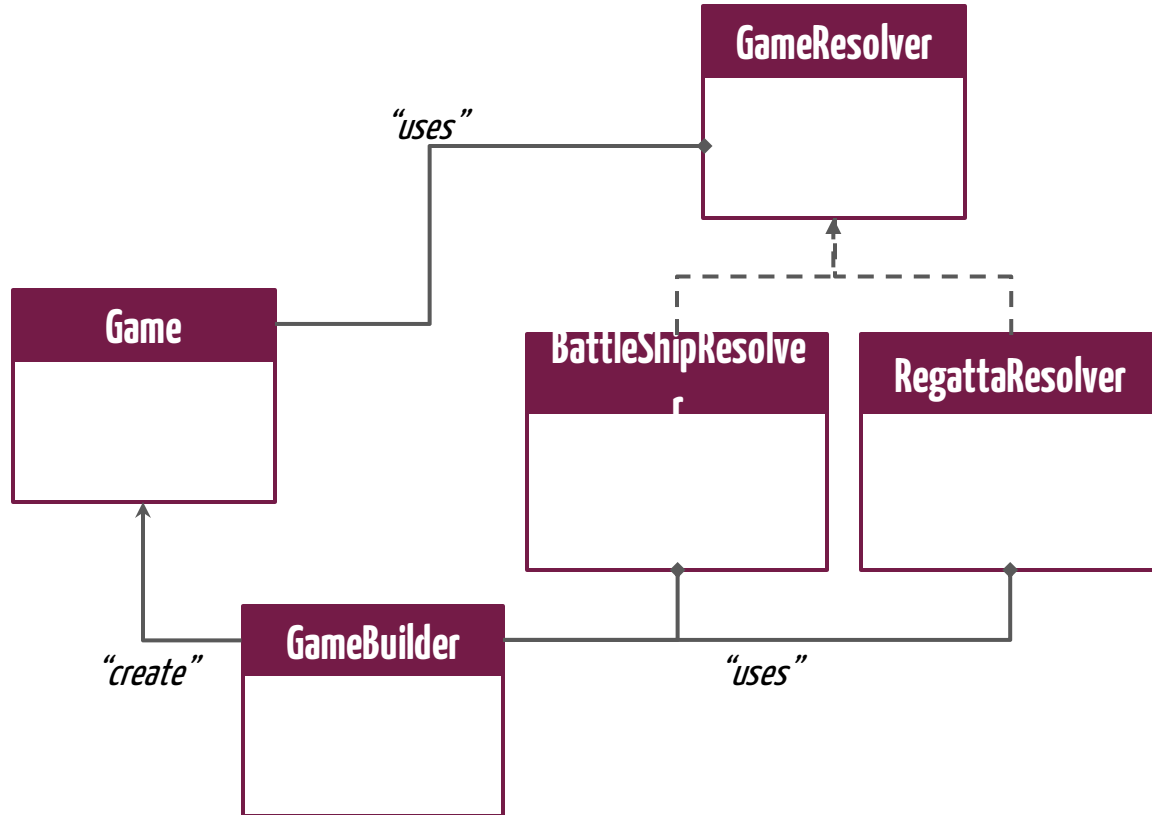


# Resolution

```
public class Game {  
    private static Logger logger = Logger.getLogger(Game.class);  
    private GameResolver resolver;  
    private GameGoal goal;  
  
    private RoundMovementsRunner roundMovementsRunner;  
    private Sea sea;  
    private List<Player> players;  
  
    private int maxRounds;  
    private int currentRound;
```

Change for a “high level” class instead.

# Resolution



**Q&A**



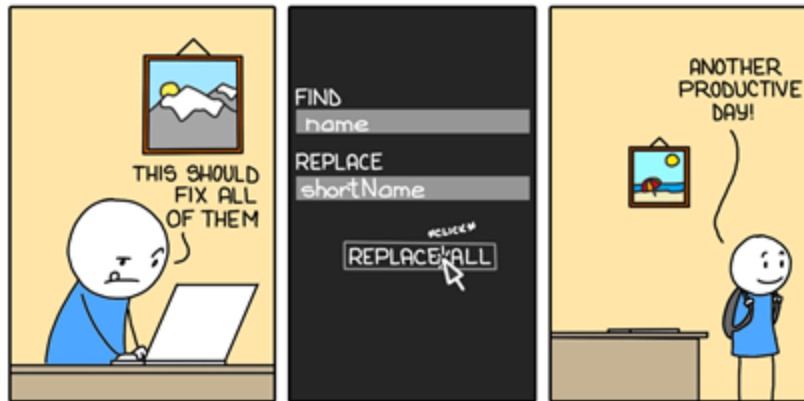
# 1



**Refactoring**

# Renamings stuff

REPLACE ALL



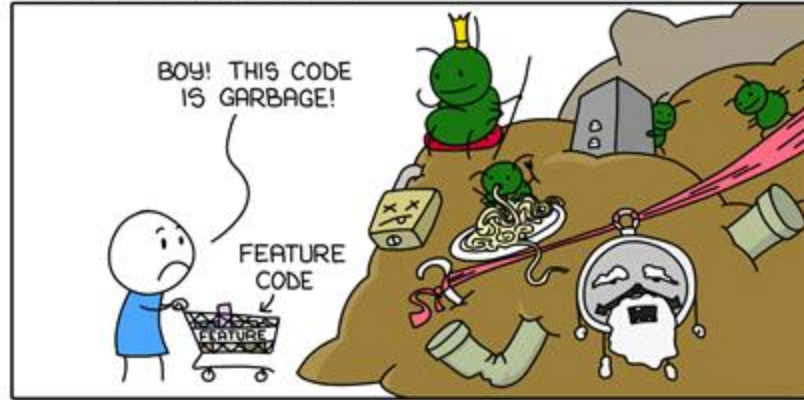
MONKEYUSER.COM

# Renamings stuff

**Your IDE reads and  
understands your code, you  
should use its power!**

# Clean-up your room!

CODE ENTROPY



MONKEYUSER.COM



# Clean-up your room!

**Add, rename, move folders  
along with code addition to  
your project...**

# Avoid code duplication

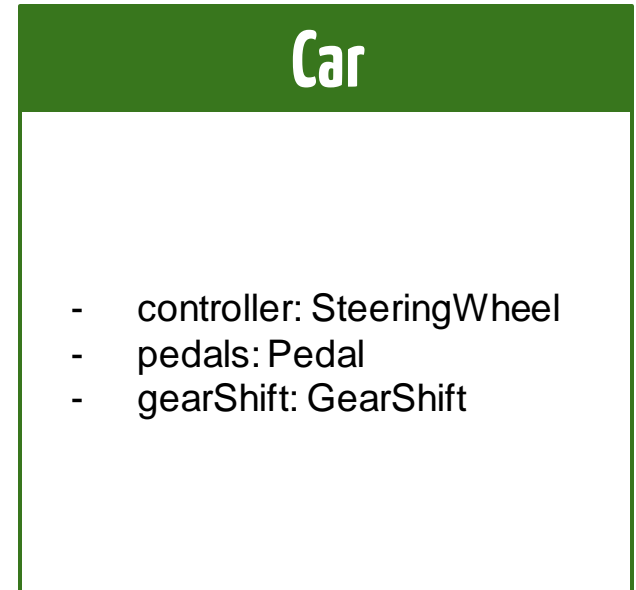
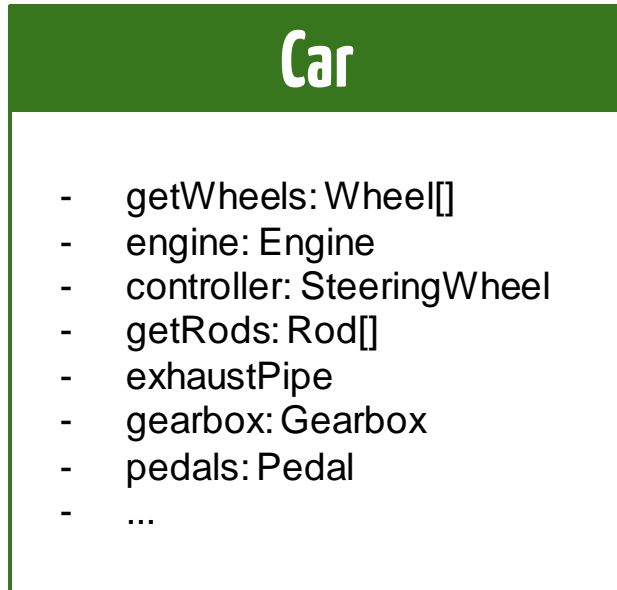




## Car

- getWheels: Wheel[]
- engine: Engine
- controller: SteeringWheel
- getRods: Rod[]
- exhaustPipe
- gearbox: Gearbox
- pedals: Pedal
- ...

# Only expose a useful interface



# Only expose a useful interface

**What is true for *classes* is  
also true for *packages*...**

# 2



**Keep in mind**

# KISS

**Keep  
It**

**Simple,  
Stupid!**

**DRY**

**Don't  
Repeat  
Yourself**



**YAGNI**

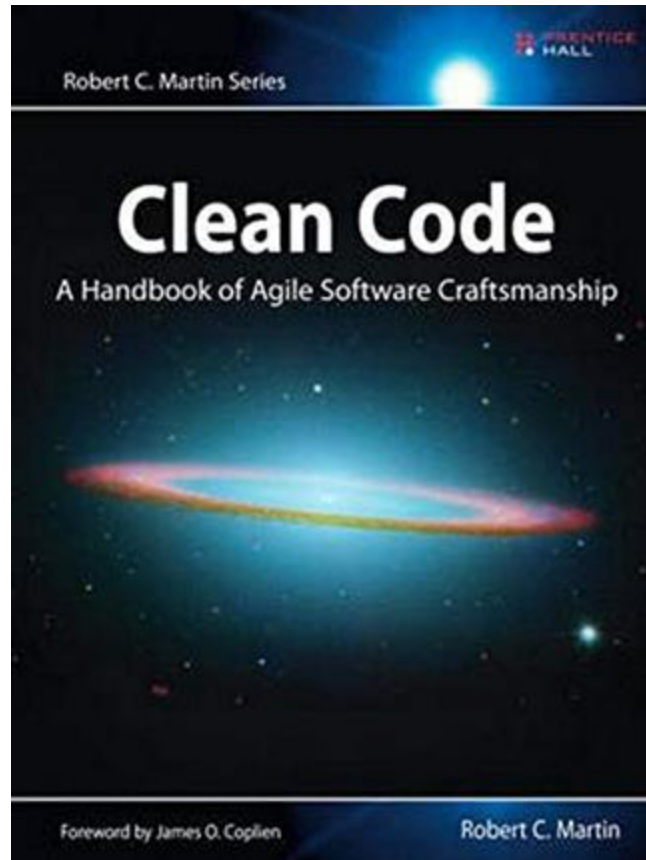
**You aren't  
gonna  
need it!**

# 3



**Clean code**

# Clean Code: reference



# Code readability

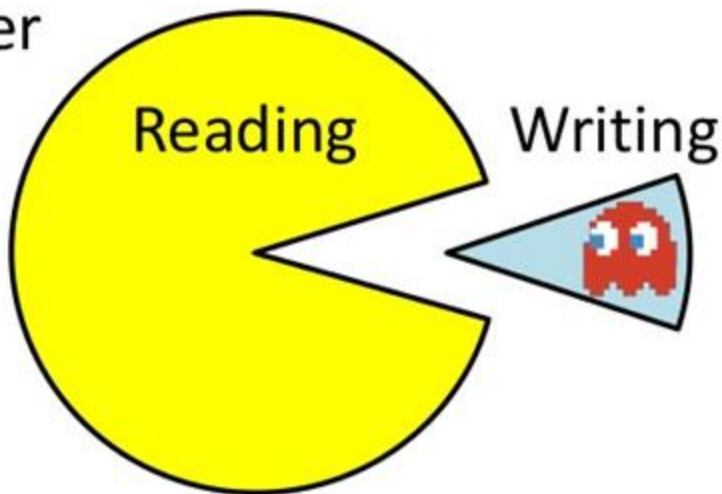
“The ratio of time spent

**reading vs. writing**

is well over

**10:1”**

– Clean Code



Stanford University

# Naming

bigButt

# Naming

**bigButt → bigButton**

Don't shorten variable names...

# Naming

bigButt → bigButton → confirmButton

Keep it functional...

# Don't mix apples and oranges

**Each lines of your function  
should be directly related to  
your function**



# Don't mix apples and oranges

```
public void runGame(String gameld) {  
    Game game = new Game();  
    game.setName(gameld.split(".")[0]);  
    GameStatus status = game.run();  
    new FilePrinter(new File(gameld + ".txt")).print(status.toString());  
}
```

# Don't mix apples and oranges

```
public void runGame(String gameld) {  
    Game game = new Game();  
    game.setName(gameld.split(".")[0]);  
    GameStatus status = game.run();  
    new FilePrinter(new File(gameld + ".txt")).print(status.toString());  
}
```

String splitting...

# Don't mix apples and oranges

```
public void runGame(String gameld) {  
    Game game = new Game();  
    game.setName(gameld.split(".")[0]);  
    GameStatus status = game.run();  
    new FilePrinter(new File(gameld + ".txt")).print(status.toString());  
}
```

Array data access...

# Don't mix apples and oranges

```
public void runGame(String gameld) {  
    Game game = new Game();  
    game.setName(gameld.split(".")[0]);  
    GameStatus status = game.run();  
    new FilePrinter(new File(gameld + ".txt")).print(status.toString());  
}
```

String concatenation...

# Don't mix apples and oranges

```
public void runGame(String gameld) {  
    Game game = new Game();  
    game.setName(gameld.split(".")[0]);  
    GameStatus status = game.run();  
    new FilePrinter(new File(gameld + ".txt")).print(status.toString());  
}
```

File system access...

# Don't mix apples and oranges

```
public void runGame(String gameId) {  
    Game game = new Game();  
    game.setName(generateGameNameFromId(gameId));  
    GameStatus status = game.run();  
    printGameStatus(status);  
}
```

# Keep your methods short

Explain what your function is doing, then code...

# Keep your methods short

10 lines of code should be enough...



# 4

A close-up photograph of a dartboard. The board has a wooden face with black and white segments. A red and white checkered dart is stuck in the center bullseye. The bullseye is a small red circle in the center of a larger blue circle. The number 15 is visible on the right side of the board.

**Let's refactor some code**

```

public double calculateAngle(JsonNode boatPosition, Position position) {
    double orientationBoat = boatPosition.get("orientation").doubleValue();
    double x = position.getX();
    double y = position.getY();

    double[] vectorBoat = new double[2];
    vectorBoat[0] = Math.cos(orientationBoat);
    vectorBoat[1] = Math.sin(orientationBoat);

    double[] vectorDirection = new double[2];
    vectorDirection[0] = x - boatPosition.get("x").doubleValue();
    vectorDirection[1] = y - boatPosition.get("y").doubleValue();

    double normeBoat = Math.sqrt(Math.pow(vectorBoat[0], 2) + Math.pow(vectorBoat[1], 2));
    double normeDirection = Math.sqrt(Math.pow(vectorDirection[0], 2) + Math.pow(vectorDirection[1], 2));
    double scalaire = vectorBoat[0] * vectorDirection[0] + vectorBoat[1] * vectorDirection[1];

    double angle = Math.acos(scalaire / (normeBoat * normeDirection));

    if (vectorDirection[1] - vectorBoat[1] < 0) {
        angle = -angle;
    }

    return angle;
}

```

```
public double calculateAngle(Point boatPosition, Point position) {  
    return Vector.fromPoints(boatPosition, position).angle();  
}
```

**Q&A**