EMBEDDED ACADEMY

★ PEDAL TO THE MEDAL ★

BOSCH

BGSV Embedded Academy (BEA)

Focused Program to Develop Embedded Competence

BGSV EMBEDDED ACADEMY

Technical Competence

- T1: Automotive Basics (Sensor, SW, Mobility Solution)
- T2: Automotive SW Architecture (AUTOSAR)
- T3: Embedded Programming
- T5: Test Overview

Methodological Competence

- M1: SW Development Lifecycle
- M3: Clean Code

Process Competence

- P1: Requirements Engineering
- P2: Design Principles
- P3: Review
- P4: Safety & Security

Classroom training, Online Self-learning, Live Demo

Purpose: Develop basic general embedded competence

Bosch Global Software Technologies
alt_future

Disclaimer

- This slide is a part of BGSV Embedded Academy (BEA) program and only used for BEA training purposes.
- This slide is Bosch Global Software Technology Company Limited's internal property. All rights reserved, also regarding any disposal, exploitation, reproduction, editing, distribution as well as in the event of applications for industrial property rights.
- This slide has some copyright images and text, which belong to the respective organizations.

M3
CLEAN CODE

Agenda

1. Writing Code
2. Code Design
3. Clean Code

WRITING CODE

Writing Code
Agenda

- ✓ Naming conventions
- ✓ Functions
- ✓ Comments
- ✓ Formatting

Naming conventions
Basic rule

- ✓ Classes and Objects — **Noun** — **Noun phrase**
- ✓ Variables, Properties, Parameters — **Noun**
- ✓ Methods and Functions — **Verb** — **Verb phrase**

Naming conventions
Use Intention-Revealing Names

- ✓ Names should reveal intent
- ✓ Don't name a variable, class, or method which needs some explanation in comments

```
int d; // elapsed time in days
```

→

```
int elapsedTimeInDays;
```

- ✓ Should tell: **WHY** it exists, **WHAT** it does, and **HOW** it is used

```
for (int j=0; j<34; j++) {
    s += (t[j]*4)/5;
}
```

→

```
int realDaysPerIdealDay = 4;
const int WORK_DAYS_PER_WEEK = 5;
int sum = 0;
for (int j=0; j < NUMBER_OF_TASKS; j++) {
    int realTaskDays = taskEstimate[j] * realDaysPerIdealDay;
    int realTaskWeeks = (realdays / WORK_DAYS_PER_WEEK);
    sum += realTaskWeeks;
}
```

Naming Conventions
Avoid Disinformation

✓ Avoid leaving false clues that obscure the meaning of code

```
public List        sparkPlugs
public Collection  sparkPlugList
public SparkPlug   sparkPlug
```

→

```
public List        sparkPluglist
public Collection  sparkPlugCollection
public SparkPlug   currentSparkPlug
```

Naming Conventions
Make Meaningful Distinctions

- ✓ If something mean different, then the names must be different
- ✓ Avoid using noise word such as “ProductInfo” or “ProductData”

```
public String   productInfo
public String   productDetails
```

→

```
public String   productName
public String   productDesciption
```

Naming Conventions
Use Pronounceable And Searchable Names

- ✓ Make your names pronounceable
- ✓ Make your names Searchable
- ✓ Avoid encoding

```
class DtaRcrd102 {
    private Date genymdhms;
    private Date modymdhms;
    private final String pszqint = "102";
};
```

→

```
class Customer {
    private Date generationTimestamp;
    private Date modificationTimestamp;
    private final String recordId = "102";
};
```

Bosch Global Software Technologies
alt_future

Naming Conventions
Take away

Choosing good names takes time but **saves** more than it takes.

✓ Classes and Objects	Noun	Noun phrase
✓ Variables, Properties, Parameters	Noun	
✓ Methods and Functions	Verb	Verb phrase

- ✓ Intention-**Revealing** Names
- ✓ **Avoid** Disinformation
- ✓ Meaningful **Distinctions**
- ✓ Pronounceable
- ✓ Searchable

Bosch Global Software Technologies
alt_future

Functions
Small

- ✓ **Small** and should be **smaller**
- ✓ Indent level of a function should not be greater than one or two

```
public static String renderPageWithSetupsAndTeardowns(
    PageData pageData, boolean isSuite) throws Exception {
    if (isTestPage(pageData))
        includeSetupAndTeardownPages(pageData, isSuite);
    return pageData.getHtml();
}
```

1 2

Functions
Do One Thing

✓ Should do one thing, do it well, do it only

```
public static boolean isCurrentUserInRole(String authority) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    Stream<String> authorities = authentication.getAuthorities().stream().map(GrantedAuthority::getAuthority);
    return authentication != null &&
           authorities.anyMatch(authority::equals);
}
```

```
public static boolean isCurrentUserInRole(String authority) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    return authentication != null &&
        getAuthorities(authentication).anyMatch(authority::equals);
}

private static Stream<String> getAuthorities(Authentication authentication) {
    return authentication.getAuthorities().stream()
        .map(GrantedAuthority::getAuthority);
}
```


alt_future

Functions
One Level of Abstraction per Function

- ✓ Statements within our function are all at the same level of abstraction
- ✓ Read code from top to bottom

Abstraction Level 1

Abstraction Level 2

```
public static void main(String[] args) {
    Application app = new Application(EtravelApp.class);
    DefaultProfileUtil.addDefaultProfile(app);
    Environment env = app.run(args).getEnvironment();
    logApplicationStartup(env);
}

public static void addDefaultProfile(Application app) {
    Map<String, Object> defProperties = new HashMap<>();
    defProperties.put(APP_PROFILE_DEFAULT, AppConstants.APP_PROFILE_DEVELOPMENT);
    app.setDefaultProperties(defProperties);
}
```

Abstraction Level 3

Functions
“One Switch“ rule

✓ Should use **polymorphism** to keep the independency between logic and its implementation

```
public Money calculatePay(Employee e)
        throws InvalidEmployeeType {
    switch (e.type) {
        case COMMISSIONED:
            return calculateCommissionedPay(e);
        case HOURLY:
            return calculateHourlyPay(e);
        case SALARIED:
            return calculateSalariedPay(e);
        default:
            throw new InvalidEmployeeType(e.type);
    }
}
```

```
public abstract class Employee {
    public abstract boolean isPayday();
    public abstract Money calculatePay();
    public abstract void deliverPay(Money pay);
}
public interface EmployeeFactory {
    public Employee makeEmployee(EmployeeRecord r)
            throws InvalidEmployeeType;
}
public class EmployeeFactoryImpl implements EmployeeFactory
{
    public Employee makeEmployee(EmployeeRecord r)
            throws InvalidEmployeeType {
        switch (r.type) {
            case COMMISSIONED:
                return new CommissionedEmployee(r) ;
            case HOURLY:
                return new HourlyEmployee(r);
            case SALARIED:
                return new SalariedEmploye(r);
            default:
                throw new InvalidEmployeeType(r.type);
        }
    }
}
```

Functions
Function Arguments

- ✓ The ideal number of arguments for a function is zero
- ✓ Next comes one, followed closely by two
- ✓ Three arguments should be avoided where possible
- ✓ More than three requires very special justification.

Likely that those arguments should be wrapped into a class of their own

Keep the number as less as possible

Functions
Have No Side Effects

✓ Not does other *hidden* things

```
public class UserValidator {
    private Cryptographer cryptographer;

    public boolean checkPassword(String userName, String password) {
        User user = UserGateway.findByName(userName);
        if (user != User.NULL) {
            String codedPhrase = user.getPhraseEncodedByPassword();
            String phrase = cryptographer.decrypt(codedPhrase, password);
            if ("Valid Password".equals(phrase)) {
                Session.initialize();
                return true;
            }
        }
        return false;
    }
}
```

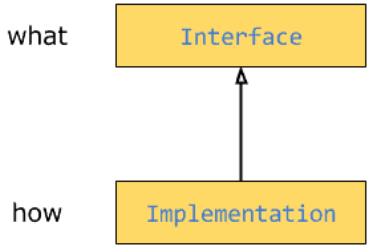
Functions
Don’t repeat yourself (DRY)

- ✓ Don’t copy and paste the same code over and over again. Consider the **abstraction**!
- ✓ Same algorithm but different codes is still a **duplication**.

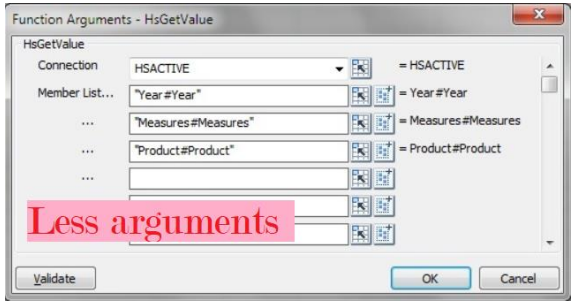
```
1
2   public class CustomerNameChanger {
3       public void ChangeName(CustomerDbContext context, int customerID, string name) {
4           var customer = context.Customer.SingleOrDefault(x => x.customerID == customerID);
5           if(customer == null)
6               throws new Exception(string.Format("Customer {0} was not found.", customerID));
7
8               customer.Name = name;
9       }
10  }
11
12  public class CustomerAddressChanger {
13      public void ChangeAddress(CustomerDbContext context, int customerID, string address,
14                                string postalCode, string city) {
15          var customer = context.Customer.SingleOrDefault(x => x.customerID == customerID);
16          if(customer == null)
17              throws new Exception(string.Format("Customer {0} was not found.", customerID));
18
19              customer.Address = address;
20              customer.PostalCode = postalCode;
21              customer.City = city;
22      }
23  }
```

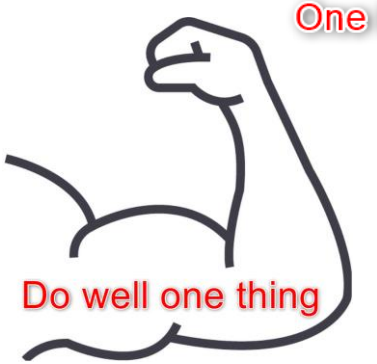
Bosch Global Software Technologies
alt_future

Functions
Take away



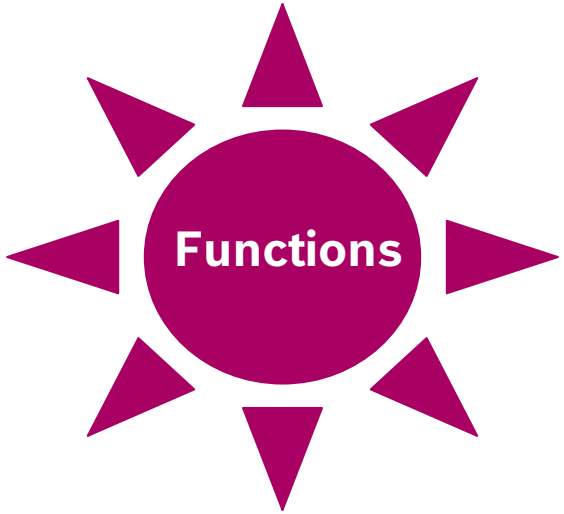
One level Abstraction

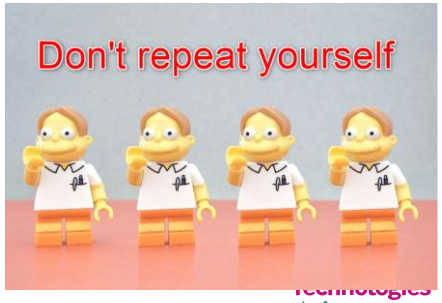




Do well one thing

Small and Smaller





Comments

*"Don't comment **bad code***

*— **rewrite** it."*

Bosch Global Software Technologies
alt_future

Comments
Pros…

```
1
2   private final String HTTP_DATE_REGEXP =
3   "[SMTWF][a-z]{2}\\,\\s[0-9]{2}\\s[JFMASOND][a-z]{2}\\s"+
4   "[0-9]{4}\\s[0-9]{2}\\:[0-9]{2}\\:[0-9]{2}\\sGMT";
5
6
```

```
1
2   private final String HTTP_DATE_REGEXP =
3   "[SMTWF][a-z]{2}\\,\\s[0-9]{2}\\s[JFMASOND][a-z]{2}\\s"+
4   "[0-9]{4}\\s[0-9]{2}\\:[0-9]{2}\\:[0-9]{2}\\sGMT";
5   // Example: "Tue, 02 Apr 2003 22:18:49 GMT"
6
```

Nothing can be quite so ***helpful*** as a **well-placed comment**

Bosch Global Software Technologies
alt_future

Comments
… and Cons

```
1
2   //Returns x + y or, if x or y is less than zero, throws an exception
3   public int Add(int x, int y)
4   {
5       return x + y;
6   }
7
```

```
1
2   // Utility method that returns when this.closed is true. Throws an exception
3   // if the timeout is reached.
4   public synchronized void waitForClose(final long timeoutMillis)
5   throws Exception
6   {
7       if(!closed)
8       {
9           wait(timeoutMillis);
10          if(!closed)
11          throw new Exception("MockResponseSender could not be closed");
12      }
13  }
14
```

Nothing can be quite so ***damaging*** as a **NOT up-to-date** and **Misinformation comment**

Bosch Global Software Technologies
alt_future

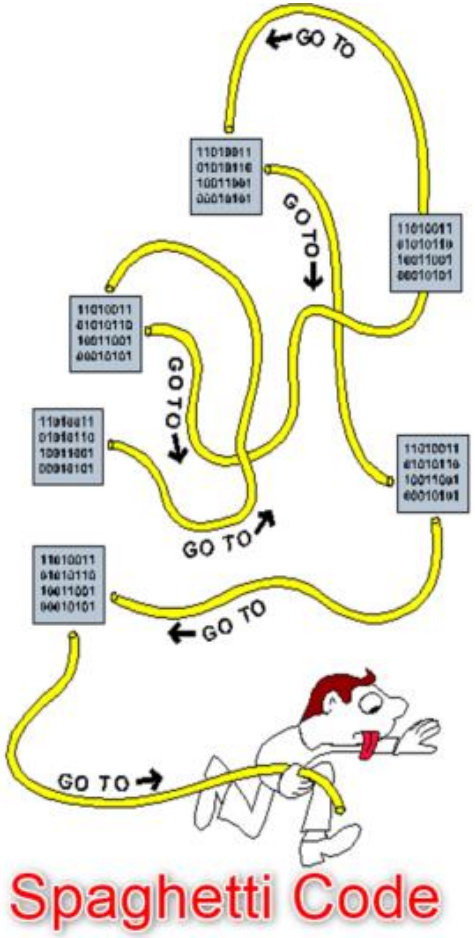
Comments
Explain yourself in Code

```
1
2
3   if (employee.isFlags() && HOURLY_FLAG && employee.getAge() > 65)
4   {
5       ...
6   }
7
```

Explain Yourself in Code

```
1
2   if (employee.isEligibleForFullBenefits())
3   {
4       ...
5   }
6
```

Comments
Comments for Bad code



Try to **clean** the code itself

Good comments

- ✓ Legal comments
- ✓ **Informative** comments
- ✓ Explanation of **Intent**
- ✓ Clarification
- ✓ Warning of **Consequences**
- ✓ **TODO** comments
- ✓ Amplification

```
// format matched kk:mm:ss EEE, MMM dd, yyyy
Pattern timeMatcher = Pattern.compile(
"\\d*:\\d*:\\d* \\w*, \\w* \\d*, \\d*");
```

```
//This is our best attempt to get a race condition
//by creating large number of threads.
for (int i = 0; i < 25000; i++) {
    ...
    thread.start();
}
```

```
public static SimpleDateFormat makeStandardHttpDateFormat()
{
    //SimpleDateFormat is not thread safe,
    //so we need to create each instance independently.
    SimpleDateFormat df = new SimpleDateFormat("EEE, dd MMM yyyy HH:mm:ss z");
    df.setTimeZone(TimeZone.getTimeZone("GMT"));
    return df;
}
```

Bad comments

- ✗ **Misleading** comments
- ✗ **Journal** comments
- ✗ **Redundant** comments
- ✗ Noise comments
- ✗ **Nonlocal** Information
- ✗ Commented-out code

```
// Changes (from 11-Oct-2001)
// --------------------------
// 11-Oct-2001 : Re-organised the class and moved it to new package
// 03-Oct-2002 : Fixed errors reported by Checkstyle;
// 13-Mar-2003 : Implemented Serializable;
// 29-May-2003 : Fixed bug in addMonths method;
```

```
function DISCOUNT (quantity, price) {
    //Calculate a discount for items greater than 100 units
    if quantity >= 100                          // Check if it's more than 100 units
        DISCOUNT = quantity * price * 0.1;      // Calculate a 10% discount
    else
        DISCOUNT = 0;                           // Otherwise no discount
    end if;

    DISCOUNT = Application.Round(DISCOUNT,2); // Round to 2 decimal places
}
```

```
// Port on which fitnesse would run. Defaults to <b>8082</b>.
//     @param fitnessePort
public void setFitnessePort(int fitnessePort) {
    this.fitnessePort = fitnessePort;
}
```

Comments
Take away

Rather than spend your time writing the **comments** that explain the mess you've made, spend it **cleaning** that mess.

Guru programmers tell you **why** other implementations were **not** chosen.

Great programmers tell you **why** a particular implementation was chosen.

Good programmers **comment** their code.

Bosch Global Software Technologies
alt_future

FORMATTING

The **broken window** theory

Global Software Technologies
alt_future

Power of formatting

Hmm... No one cares about how to write an easy-to-read code here. The other modules of this project will be like this only!!!

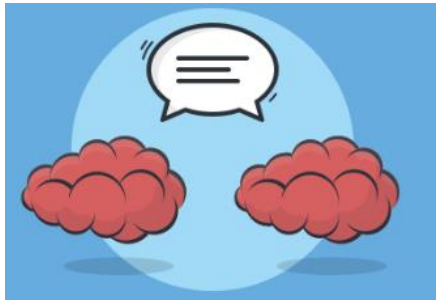
```
function addlist(i){
document                                    tener("click",

                                            lor = "green
                                            -checked","t

                                            yle.left.substri
                                            1].style.left.substr
                                            ked='true']")[0].style.top.substri
  var  y2 = $("[data-checked='true']")[1].style.top.substr
  console.log(`${x1} ${y1} ${x2} ${y2}`)
var c = document.getElementById("canvas");
  var ctx = c.getContext("2d");
ctx.beginPath();
ctx.moveTo(x1,y1);
ctx.lineTo(x2, y2);
ctx.stroke();}
```

Power of formatting

- Code formatting is about **communication**

  … and communication is ***first need*** of

  a professional developer



- The code itself might be changed in the future

  … but the ***coding style*** and ***readability***

  will ***live*** together with the project.

```
function register()
{
    if (!empty($_POST)) {
        $msg = '';
        if ($_POST['user_name']) {
            if ($_POST['user_password_new']) {
                if ($_POST['user_password_new'] === $_POST['user_password_repeat']) {
                    if (strlen($_POST['user_password_new']) > 5) {
                        if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
                            if (preg_match('/^[a-z\d]{2,64}$/i', $_POST['user_name'])) {
                                $user = read_user($_POST['user_name']);
                                if (!isset($user['user_name'])) {
                                    if ($_POST['user_email']) {
                                        if (strlen($_POST['user_email']) < 65) {
                                            if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                                                create_user();
                                                $_SESSION['msg'] = 'You are now registered so please login';
                                                header('Location: ' . $_SERVER['PHP_SELF']);
                                                exit();
                                            } else $msg = 'You must provide a valid email address';
                                        } else $msg = 'Email must be less than 64 characters';
                                    } else $msg = 'Email cannot be empty';
                                } else $msg = 'Username already exists';
                            } else $msg = 'Username must be only a-z, A-Z, 0-9';
                        } else $msg = 'Username must be between 2 and 64 characters';
                    } else $msg = 'Password must be at least 6 characters';
                } else $msg = 'Passwords do not match';
            } else $msg = 'Empty Password';
        } else $msg = 'Empty Username';
        $_SESSION['msg'] = $msg;
    }
    return register_form();
}
```


Bosch Global Software Technologies
alt_future

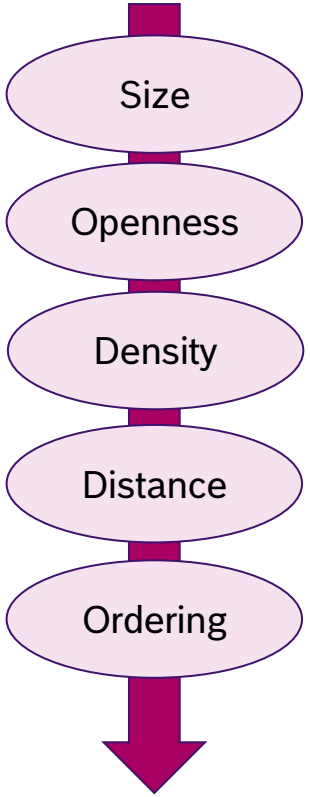
Power of formatting

- Define the TEAM RULE (Coding standard), pay attention to details and keep it is done consistently to prevent “ the broken window” to happen

Bosch Global Software Technologies
alt_future

Formatting

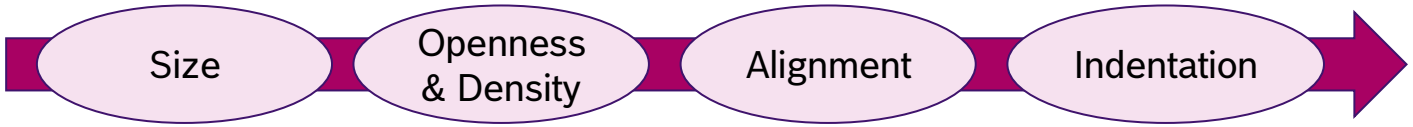
Vertical Formatting




```python
def generate_sequences(n, length=3, prop_no_change=0.2):
    """Short summary.

    Parameters
    ----------
    n : type
        Description of parameter `n`.
    length : type
        Description of parameter `length`.
    prop_no_change : type
        Description of parameter `prop_no_change`.

    Returns
    -------
    type
        Description of returned object.

    """
    x_range = [0, lc_all.width - 1]
    y_range = [0, lc_all.height - 1]
    all_samples = []
    max_no_change = round(n * prop_no_change)
    max_changed = n - max_no_change
    n_changed = 0
    n_no_change = 0
    while len(all_samples) < n:
        while True:
            x = random.randrange(*x_range)
            y = random.randrange(*y_range)
            win = Window(x, y,
            which = range(lc_a
```

randrange(self, start, stop=None, step=1, _int=int)

Choose a random item from range(start, stop[, step])



Horizontal Formatting

Vertical Formatting

- ✓ Vertical **Size**: How big should a source file be?

  → Typically 200 lines long, upper limit of 500

- ✓ Vertical **Openness** between concepts

Add blank line

```java

package fitnesse.wikitext.widgets;

import java.util.regex.*;

public class BoldWidget extends ParentWidget {
    public static final String REGEXP = "'''.+?'''";
    private static final Pattern pattern = Pattern.compile("'''(.+?)'''",
        Pattern.MULTILINE + Pattern.DOTALL
    );

    public BoldWidget(ParentWidget parent, String text) throws Exception {
        super(parent);
        Matcher match = pattern.matcher(text);
        match.find();
        addChildWidgets(match.group(1));
    }

    public String render() throws Exception {
        StringBuffer html = new StringBuffer("<b>");
        html.append(childHtml()).append("</b>");
        return html.toString();
    }
}
```

Vertical Formatting

- ✓ Vertical **Size**: How big should a source file be?

  → Typically 200 lines long, upper limit of 500

- ✓ Vertical **Openness** between concepts
- ✓ Vertical **Density** within concept

Keep dense

```java

package fitnesse.wikitext.widgets;

import java.util.regex.*;

public class BoldWidget extends ParentWidget {
    public static final String REGEXP = "'''.+?'''";
    private static final Pattern pattern = Pattern.compile("'''(.+?)'''",
        Pattern.MULTILINE + Pattern.DOTALL
    );

    public BoldWidget(ParentWidget parent, String text) throws Exception {
        super(parent);
        Matcher match = pattern.matcher(text);
        match.find();
        addChildWidgets(match.group(1));
    }

    public String render() throws Exception {
        StringBuffer html = new StringBuffer("<b>");
        html.append(childHtml()).append("</b>");
        return html.toString();
    }
}
```

Vertical Formatting

- ✓ Vertical **Size**: How big should a source file be?

  → Typically 200 lines long, upper limit of 500

- ✓ Vertical **Openness** between concepts
- ✓ Vertical **Density** within concept
- ✓ Vertical **Distance**: Related concepts should be closed to each other
- ✓ Vertical **Ordering**: The called function should be kept before/after the calling function

Horizontal Formatting

- ✓ Horizontal **Size**: How wide should a line be?

  → Typically 80 ~ 120 chars/line, **NEVER** have to scroll to the right

- ✓ Horizontal **Openness**

Add space

```
1
2   public class Quadratic {
3       public static double root1(double a, double b, double c) {
4           double determinant = determinant(a, b, c);
5           return (-b + Math.sqrt(determinant)) / (2*a);
6       }
7       public static double root2(int a, int b, int c) {
8           double determinant = determinant(a, b, c);
9           return (-b - Math.sqrt(determinant)) / (2*a);
10      }
11      private static double determinant(double a, double b, double c) {
12          return b*b - 4*a*c;
13      }
14  }
```

Horizontal Formatting

- ✓ Horizontal **Size**: How wide should a line be?

  → Typically 80 ~ 120 chars/line, **NEVER** have to scroll to the right

- ✓ Horizontal **Openness**

- ✓ Horizontal **Density**

Keep dense

```
1
2   public class Quadratic {
3       public static double root1(double a, double b, double c) {
4           double determinant = determinant(a, b, c);
5           return (-b + Math.sqrt(determinant)) / (2*a);
6       }
7       public static double root2(int a, int b, int c) {
8           double determinant = determinant(a, b, c);
9           return (-b - Math.sqrt(determinant)) / (2*a);
10      }
11      private static double determinant(double a, double b, double c) {
12          return b*b - 4*a*c;
13      }
14  }
```

Horizontal Formatting

✓ Horizontal **Alignment**

```java

public class FitNesseExpediter implements ResponseSender
{
    private     Socket          socket;
    private     InputStream     input;
    private     OutputStream    output;
    private     Request         request;
    private     Response        response;
    private     FitNesseContext context;
    protected   long            requestParsingTimeLimit;
    private     long            requestProgress;
    private     long            requestParsingDeadline;
    private     boolean         hasError;

    public FitNesseExpediter(Socket          s,
                             FitNesseContext context) throws Exception
    {
        this.context            = context;
        socket                  = s;
        input                   = s.getInputStream();
        output                  = s.getOutputStream();
        requestParsingTimeLimit = 10000;
    }
```

```java

public class FitNesseExpediter implements ResponseSender
{
    private Socket socket;
    private InputStream input;
    private OutputStream output;
    private Request request;
    private Response response;
    private FitNesseContext context;
    protected long requestParsingTimeLimit;
    private long requestProgress;
    private long requestParsingDeadline;
    private boolean hasError;

    public FitNesseExpediter(Socket s, FitNesseContext context) throws Exception
    {
        this.context = context;
        socket = s;
        input = s.getInputStream();
        output = s.getOutputStream();
        requestParsingTimeLimit = 10000;
    }
```

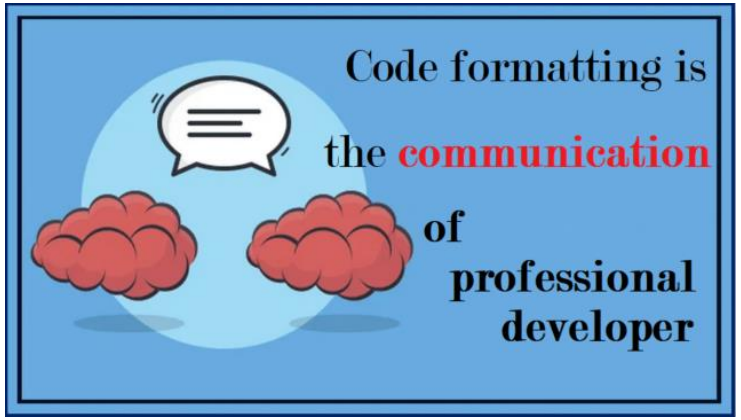
Horizontal Formatting

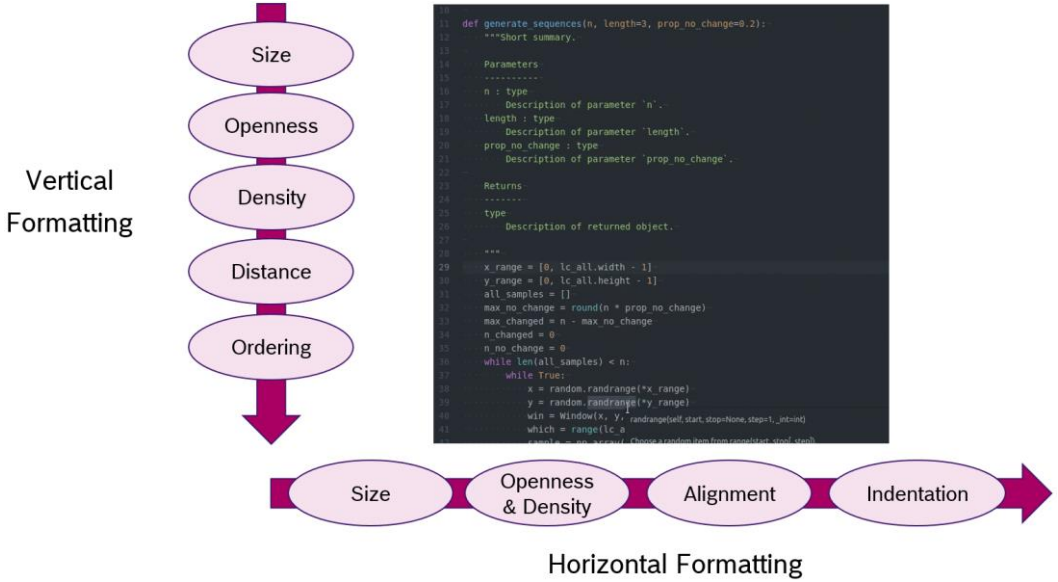
✓ Indentation

```
1
2   public class FitNesseServer implements SocketServer { private FitNesseContext
3   context; public FitNesseServer(FitNesseContext context) { this.context =
4   context; } public void serve(Socket s) { serve(s, 10000); } public void
5   serve(Socket s, long requestTimeout) { try { FitNesseExpediter sender = new
6   FitNesseExpediter(s, context);
7   sender.setRequestParsingTimeLimit(requestTimeout); sender.start(); }
8   catch(Exception e) { e.printStackTrace(); } } }
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
```

```
1
2   public class FitNesseServer implements SocketServer {
3       private FitNesseContext context;
4
5       public FitNesseServer(FitNesseContext context) {
6           this.context = context;
7       }
8
9       public void serve(Socket s) {
10          serve(s, 10000);
11      }
12
13      public void serve(Socket s, long requestTimeout) {
14          try {
15              FitNesseExpediter sender = new FitNesseExpediter(s, context);
16              sender.setRequestParsingTimeLimit(requestTimeout);
17              sender.start();
18          }
19          catch (Exception e) {
20              e.printStackTrace();
21          }
22      }
23  }
24
```

Formatting
Take away





Bosch
Global
Software
Technologies
alt_future

CODE DESIGN

Code design
Agenda

- ✓ Abstraction and Encapsulation
- ✓ Object and Data Structure
- ✓ Class
- ✓ Error handling
- ✓ Boundaries
- ✓ Unit tests

Code design

Abstraction and Encapsulation

- **Abstraction :** Process in which you collect or gather relevant data and remove non-relevant data
- **Encapsulation:** Process in which you wrap of functions and members in a single unit

Code design
Abstraction in C

private.c

```c
struct Contact
{
    int mobile_number;
    int home_number;
};
struct Contact * create_contact()
{
    struct Contact * some_contact;
    some_contact = malloc(sizeof(struct Contact));
    some_contact->mobile_number = 12345678;
    some_contact->home_number = 87654321;
    return( some_contact );
}
void delete_contact( struct Contact * some_contact )
{
    free(some_contact);
}
```

private.h

```c
struct Contact;

struct Contact * create_contact();

void delete_contact( struct Contact * some_contact );
```

main.c

```c
#include "private.h“
#include <stdio.h>
void main()
{
    struct Contact * Tony;
    Tony = create_contact();
    printf( "Mobile number: %d\n", Tony->mobile_number);
    delete_contact( Tony );
}
```

Code design
Encapsulation in C

Area.c

```
Class Rectangle {
Public :
          int length;
          int breadth;

          int getArea()
          {  return length * breadth;
          }

};
```

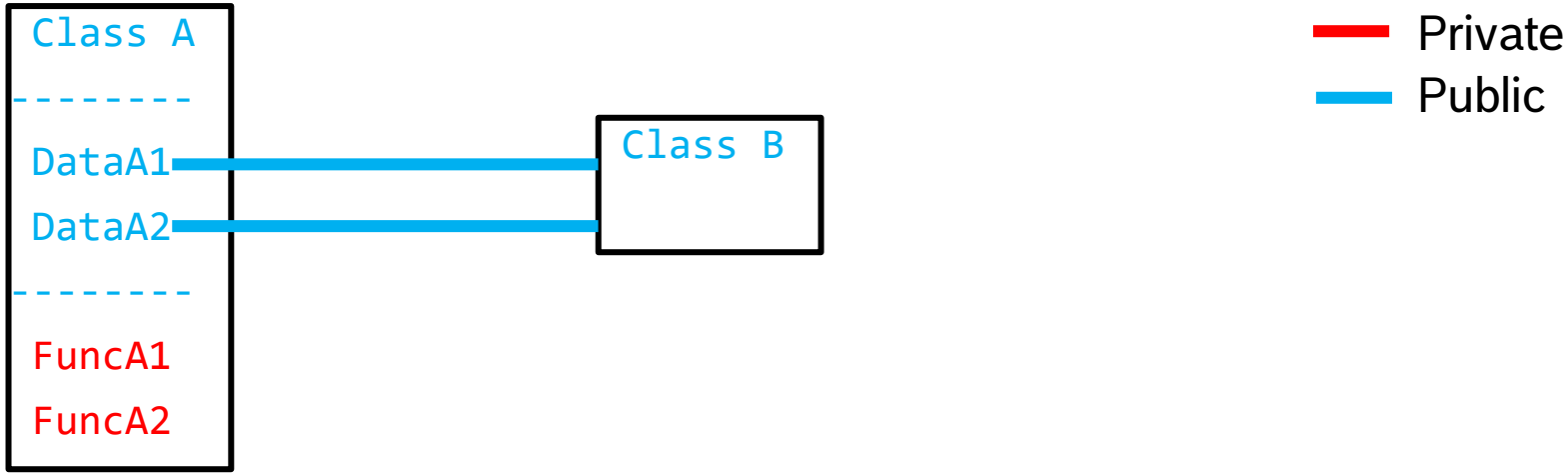
Code design
Object and Data Structure

- **Data Structure** class reveals or exposes its data (variables) and have no significant methods or functions.
- **Object Structure** class conceals their data and reveals or exposes their methods that work on those data.

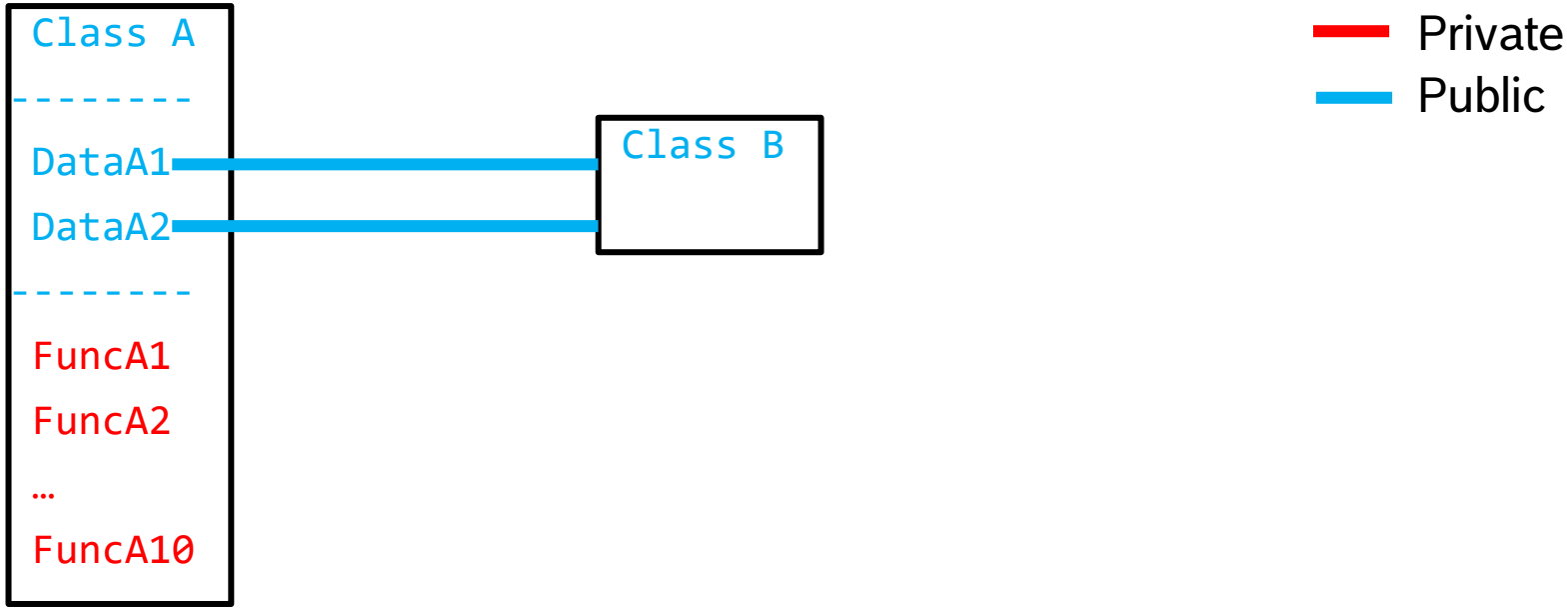
```
Class A
--------
DataA1
DataA2
--------
FuncA1
FuncA2
```

```
Class B
--------
DataB1
DataB2
--------
FuncB1
FuncB2
```

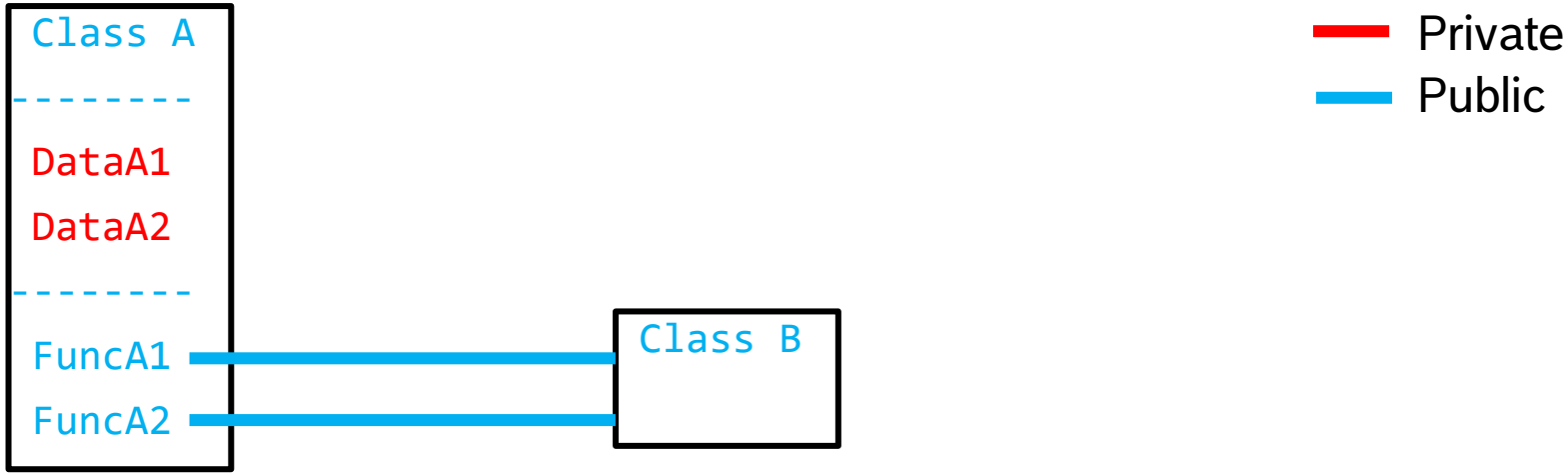
Code design
Data Structure



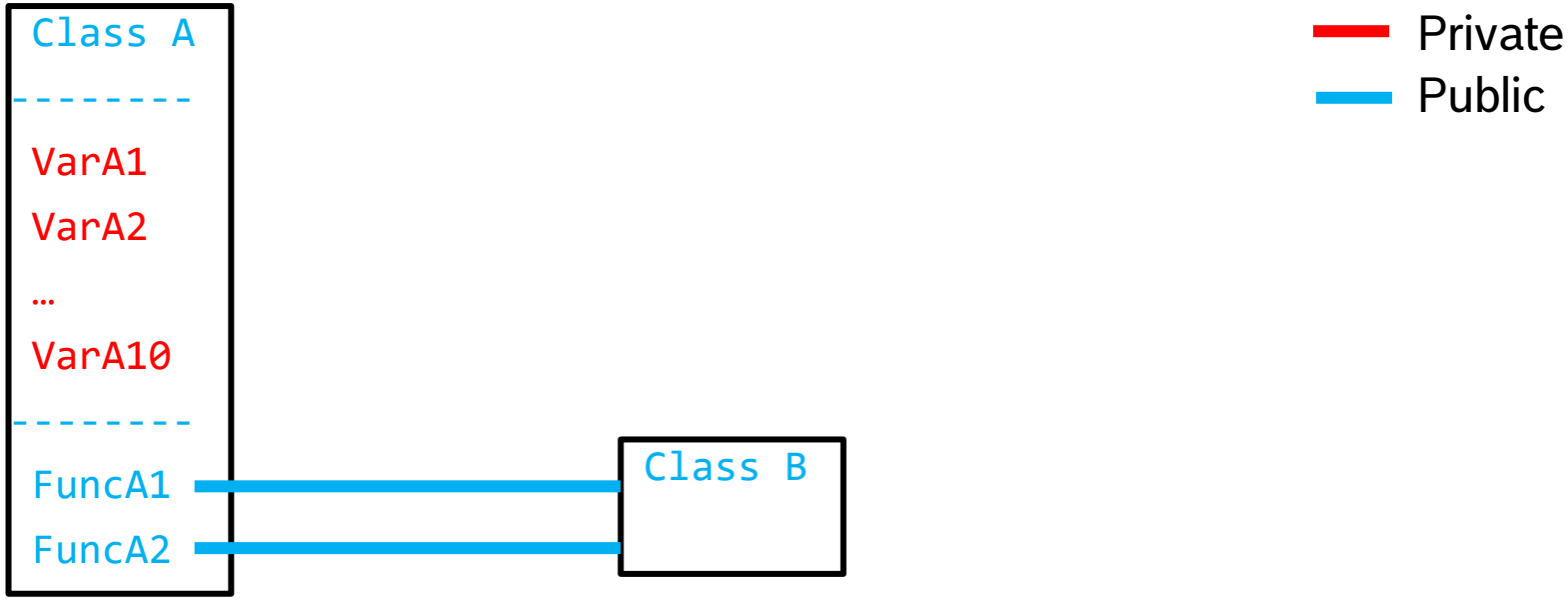
Code design
Data Structure



Code design
Object and Data Structure



Code design
Object and Data Structure



Code design
Object and Data Structure

- ✓ **Prevent exposing details of the data** instead express data in abstract terms.
- ✓ **Objects Structures** hides data behind abstractions and exposes functions.
- ✓ **Data structures** exposes data and have no significant functions.
- ✓ **Decision on choosing Objects Structure and Data Structure**.

CLASSES

Code design
Classes

Single Responsibility Principle (SRP)

A class should have one, and only one, reason to change.

Open Closed Principle (OCP)

You should be able to extend a classes behavior without modifying it.

Liskov Substitution Principle (LSP)

Derived classes must be substitutable for their base classes.

Interface Segregation Principle (ISP)

Derived classes must be substitutable for their base classes.

Dependency Inversion Principle (DIP)

Derived classes must be substitutable for their base classes.

SOLID PRINCIPLES

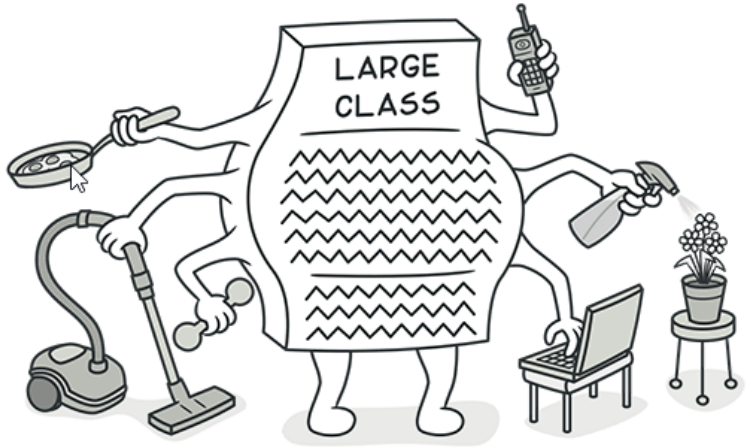
Bosch Global Software Technologies
alt_future

Code design
Classes

Classes Should Be Small!

A class can contains many fields/methods/lines of code.

Classes usually start small. But over time, they get bloated as the program grows!



Clean Code
Classes

```
public class Employee
{
    public string Name { get; set; }
    public string Address { get; set; }
    ...
    public void ComputePay() { ... }
    public void ReportHours() { ...}
}
```

Finance Team : I Want to change Condition for Payout??

Operations: I Want to change Condition for Login/Logout??

Clean Code
Classes

SRP --- Single Responsibility Principle.

Classes should have one responsibility — one reason to change!!

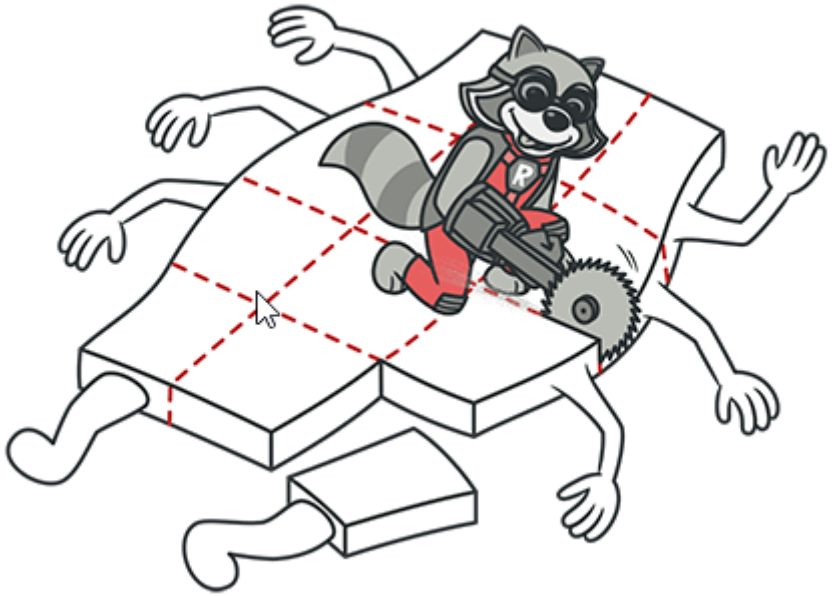
```
public class Employee
{
    public string Name { get; set; }
    public string Address { get; set; }
    ...
    public void Create() { ... }
    public void Update() { ...}
}
public class Payroll
{
    public int BaseSalary { get; set; }
    public int Allowance { get; set; }
    ...
    public void ComputePay() { ... }
}
```

Bosch Global Software Technologies
alt_future

Code design
Classes

Treatment --- **Refactoring**

- ✓ Extract Class
- ✓ Extract Subclass



Clean Code
Classes

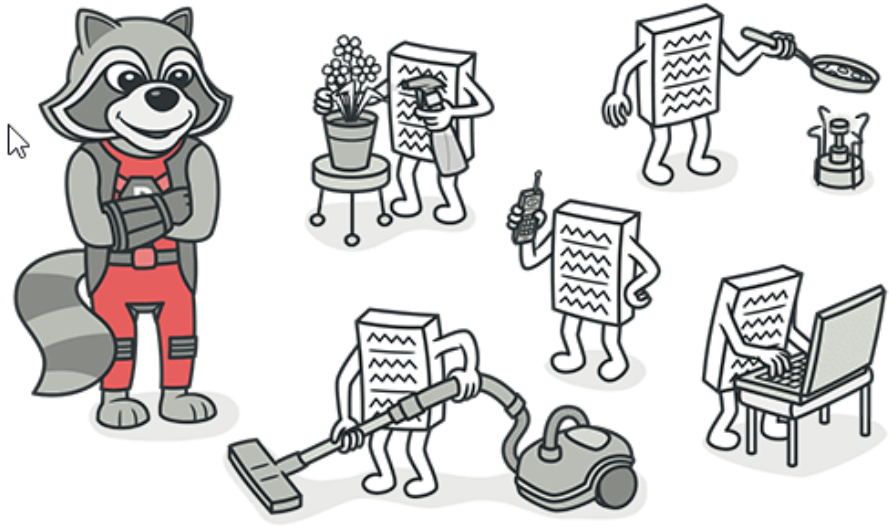
```
public class Employee
{
    public string Name { get; set; }
    public string Address { get; set; }
    ...
    public void Create() { ... }
    public void Update() { ...}
}
public class Intern : public Employee
{
    public string Name { get; set; }
    public string Address { get; set; }
    ...
    public void Create() { ... }
    public void Update() { ...}
}
```

Bosch Global Software Technologies
alt_future

Code design
Classes

Payoff

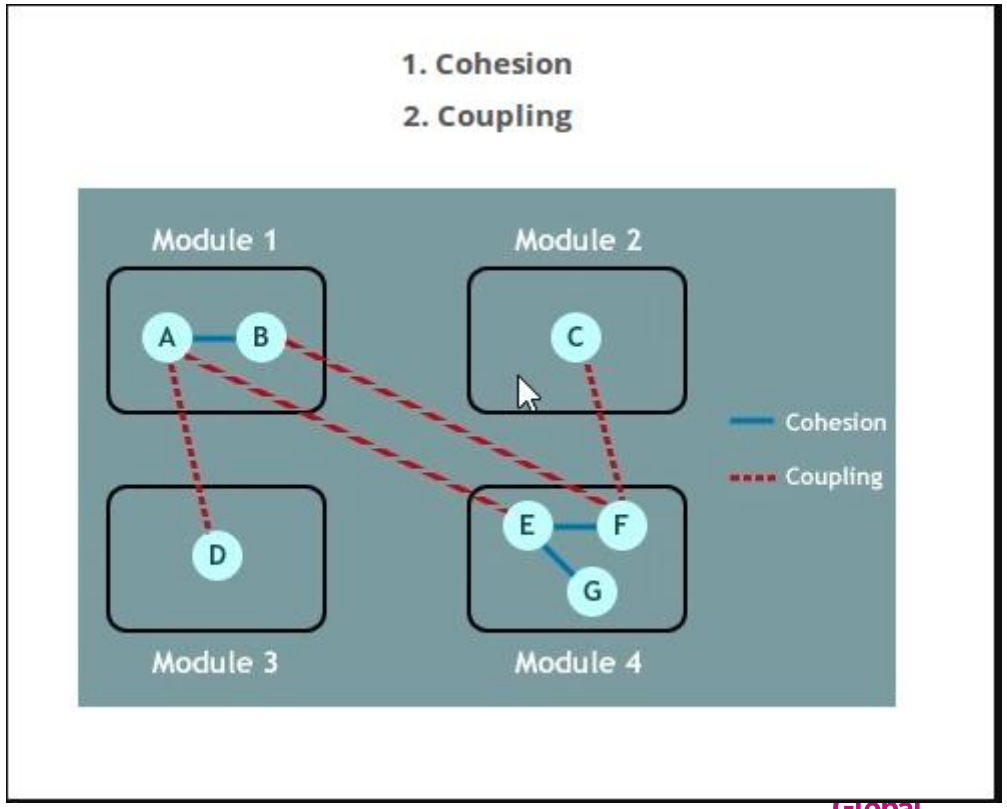
- ✓ Refactoring spares developers from needing to remember a large number of attributes for a class.
- ✓ Splitting large classes into parts avoids duplication of code and functionality.



Code design
Classes

Cohesion and Coupling

- **Cohesion-** Indication of Relationship within a module
- **Coupling** – Indication of Relationship between modules



Global Software Technologies
alt_future

Clean Code
Classes

```
-------------------
| Staff           |
-------------------
| checkEmail()    |
| sendEmail()     |
| emailValidate() |
| PrintLetter()   |
-------------------
```

```
CheckEmail ()
{
    GroupEmail();
    FilterEmail();
    FilterSpam();
    MovetoJunk();
}
```

```
SendEmail ()
{
    MailingList();
    SaveDraft();
    Send();
    DiscardDraft();
}
```

This class does a great variety of actions

Low Cohesion

Bosch Global Software Technologies
alt_future

Clean Code
Classes

```
----------------------------
| Staff                    |
----------------------------
| -salary                  |
| -emailAddr               |
----------------------------
| setSalary(newSalary)     |
| getSalary()              |
| setEmailAddr(newEmail)   |
| getEmailAddr()           |
----------------------------
```

```
SetSalary (newSalary)
{

}
```

```
getSalary ()
{

}
```

This class is focused on what it should be doing

High Cohesion

Code design
Classes

Coupling refers to how dependent two classes/modules are towards each other.

- Loose coupled classes, changing something major in one class should not affect the other.
- Tight coupling would make it difficult to change and maintain your code.

Clean Code
Classes

Good Software Design

- ✓ Should strive for **high cohesion** with little interaction with other modules of the system.

- ✓ Should strive for **loose coupling** i.e. dependency between modules should be less

ERROR HANDLING

Code design
Error Handling

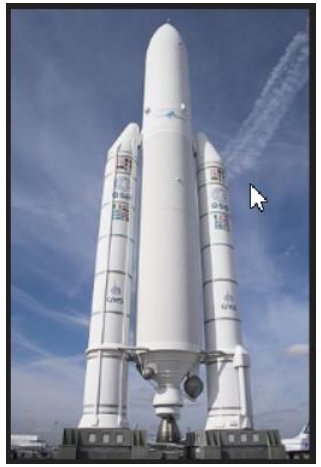
Exception is..

Abnormal or exceptional conditions requiring special processing – often changing the normal flow of program execution

Handling requires..

Specialized programming language constructs or computer hardware mechanisms.

Code design
Error Handling



Ariane 5 rocket launch failure in 1996

Issue

- Navigation system failure

Root cause

- Reused Initial reference platform SW from Ariane 4.
- Horizontal acceleration calculation caused a data conversion from a **64-bit floating point number** to a **16-bit signed integer** value to **overflow**.
- Error Handling was suppressed for performance reasons.

Impact

Start! **37 seconds** of flight. **BOOM!** 10 years and 350 million $ are turning into dust.

Bosch Global Software Technologies
alt_future

Code design
Error Handling

Error Handling is a Must and Good but when it is **not clean** ??

- **When we end up with code where only error handling can be seen.**
- **Impossible for us to find the details about the real functionality.**

Code design
Error Handling

```
function(void)
{
    if(Error_1)
    {
         /* Handle Error1*/
     }
    if(Error_2)
    {
         /* Handle Error 2*/
     }
    else
       /* Do actual function*/
}
```

Code design

Don’t Pass Null

```
void getarray(int arr[ ])
{
    printf("Elements of array are : ");
    for(int i=0;i<5;i++)
    {
        printf("%d ", arr[i]);
    }
}

int main()
{
   int arr[5]={45,67,34,78,90};
   getarray(arr);
   return 0;
}
```

getarray(Null);

getarray(arr1);

Code design

Don’t Return Null

```
int *getarray(int *a)
{

    printf("Enter the elements in an array : ");
    for(int i=0;i<5;i++)
    {
        scanf("%d", &a[i]);
    }
    return a;
}

int main()
{
  int *n;
  int a[5];
  n=getarray(a);
  printf("\nElements of array are :");
  for(int i=0;i<5;i++)
    {
        printf("%d", n[i]);
    }
    return 0;
}
```

return Null;

return a;

Code design
Error Handling

- **Clean and Robust**.
- **Error Handling should be separate from main Logic**.
- **Treat them independently which provides maintainability**.

Don’t pass or return NULL!!!

BOUNDARIES

Code design
Boundaries

Defining clean boundaries

```
if (stEventCounter_u8 > EventThreshold_C)
{
    ActivateNextEvent();
    stEventCounter_u8 = 0;
}
else
{
    stEventCounter_u8++;
}
```

Both "*stEventCounter_u8*" and "*EventThreshold_C*" are 8-bit data.

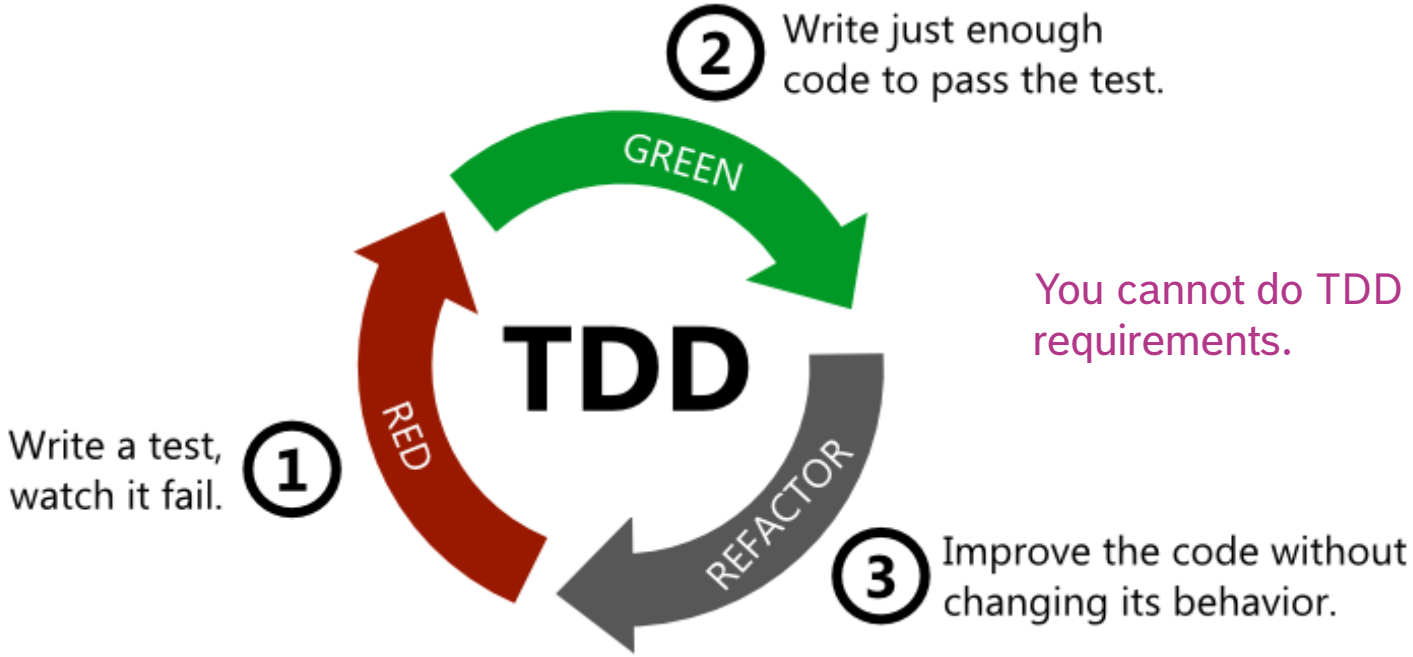
What happens if *EventThreshold_C* changes to 255??

Code design
Boundaries

- Keep boundaries **Clean** and **Separated**.

UNIT TESTS

Code design
Test Driven Development



You cannot do TDD when you don't understand the requirements.

Bosch Global Software Technologies
alt_future

Code design
Unit Tests

Why TDD??

- Easy to validate your code because you have made tests for all of it.
- Your tests describes how your code works.
- No fear to change code.

Code design
Unit Tests

Rules for Clean Tests:

- ✓ **F**ast
- ✓ **I**ndependent
- ✓ **R**epeatable
- ✓ **S**elf Validating
- ✓ **T**imely

CLEAN CODE

Clean Code

> The goal of all software-design techniques is to break a complicated problem into simple pieces

Steve McConnell

Separation of concerns

Eliminate the tight-coupling

Clean Code Principle

Clean Code

Separate construction & when using it

> Software system should separate the startup process, when the application objects are constructed and the dependencies are "wired", from the runtime logic that takes over after startup

Uncle Bob

Clean Code
Separation of Concerns

> Software systems are unique compared to physical systems. Their architectures can grow incrementally, if we maintain the proper separation of concerns

Uncle Bob

The 4 simple design rules

Simple Design Rule 1

Run All the Tests

- Systems that aren't testable aren't verifiable
- A system that cannot be verified should never be deployed

Make a testable system:

- Conform to the SRP: small and single purpose class/function
- Loose coupling and high cohesion

Clean Code
Getting Clean via Emergent Design

Simple Design Rule 2

No Duplication

duplication
- Additional work
- Additional risk
- Additional unnecessary complexity

Clean Code
Getting Clean via Emergent Design

Simple Design Rule 3

Expressive

- ***We are*** deep in an understanding of the problem we're trying to solve at the time we write code.
- ***Other maintainers*** of the code aren't going to have so deep an understanding

✓ Choosing good names, using standard nomenclature!

✓ Keeping your functions and classes small!

✓ Using well-written unit tests as documentation!

Simple Design Rule 4

Minimal Classes and Methods

this rule has the lowest priority

Our goal is to keep our overall system small while we are also keeping our functions and classes small.

Clean Code
System Emergence: Conclusion

Your simple system today can become a complex system tomorrow

Keep in mind: **Separation of Concerns**

Keep the rules: **(1) Run All the Tests**

(2) No Duplication

(3) Expressive

(4) Minimal Classes and Methods

Thank you!

Bosch
Global
Software
Technologies
alt_future