

**Eötvös Loránd Tudományegyetem
Informatikai Kar**

Szoftverttechnológia

9. előadás

Implementáció és verziókövetés

Giachetta Roberto

groberto@inf.elte.hu

<http://people.inf.elte.hu/groberto>



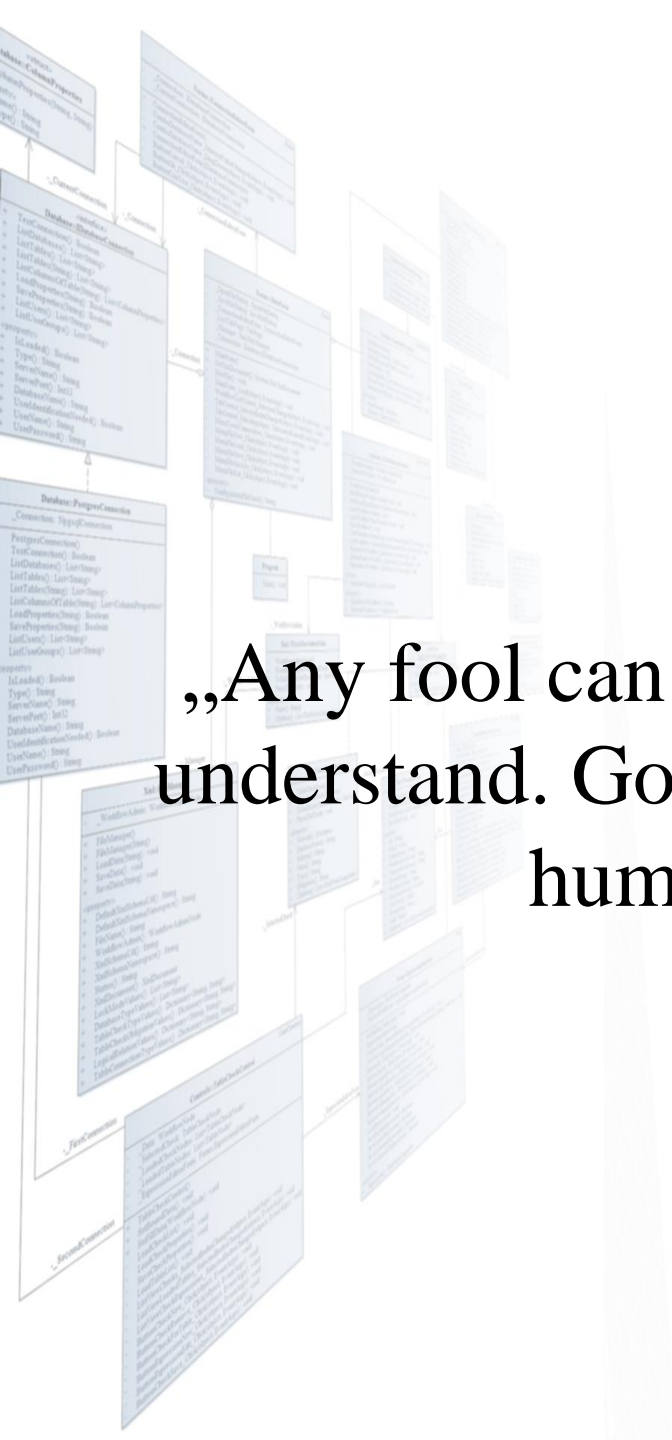


Programozási technológia 2.

Clean Code

Dr. Szendrei Rudolf
ELTE Informatikai Kar
2018.





„Any fool can write code that a computer can understand. Good programmers write code that humans can understand.”

(Martin Fowler)



Implementáció és verziókövetés

Implementáció

- Az implementáció a programkód elkészítése a rendszertervnek megfelelően adott programozási nyelven
 - az adatok megfelelő ábrázolása, reprezentálása
 - a funkciókat megvalósító algoritmusok alkalmazása
 - korábban már bevált elemek (algoritmusok, programszerkezetek) újhasznosítása
 - a minőségi mutatóknak megfelelő optimalizálások (teljesítményjavítások) végrehajtása
- Az implementációt *verifikáció* zárja, amelyben ellenőrizzük, hogy a szoftver teljesíti-e a tervben megszabott funkciókat

Implementáció és verziókövetés

Újrahasznosítás

- Az implementáció általában nagyban támaszkodik *újrahasznosításra*
 - garantálja, hogy jó, hibamentes megoldások kerüljenek alkalmazásra
 - csökkenti az implementáció (és a tesztelés) idejét és költségeit
 - az újrahasznosítás elvégezhető objektum, csomag, vagy komponens szinten
- A fejlesztő nem csupán az általa korábban fejlesztett elemeket használhatja újra, de más fejlesztő által megvalósított elemeket
 - pl. nyílt forráskódú programcsomagok
 - általában előre fordított formában állnak rendelkezésre

Implementáció és verziókövetés

Az integrált fejlesztőkörnyezet

- Az implementációhoz megfelelő *integrált fejlesztőkörnyezet (IDE)* szükséges (pl. *Eclipse*, *Visual Studio*, *Xcode*, *NetBeans*)
 - a teljes szoftver életciklust támogatja, integrálja a verziókövetést és a tesztelést
 - a fejlesztést kód-kiemeléssel (*syntax highlight*), kód-kiegészítésekkel (*code-snippet*, *intelligent code completion*), illetve kód-újratervezési eszközökkel támogatja
 - megkönnyíti külső programcsomagok integrációját (*package manager*)
 - a tesztelést nyomkövetéssel (*debugging*), egységtesztek (*unit test*), illetve teljesítményteszteléssel támogatja

Implementáció és verziókövetés

Csapatban történő implementáció

- A szoftverek általában csapatban készülnek
 - az implementáció egy egységes kódolási stílus mentén történik, egységes eszköztárral
 - minden fejlesztő csak a saját programkódján dolgozik
 - a verziókövető rendszerben általában külön fejlesztési ágban tevékenykedik
 - amennyiben más kódjában hibát talál, hibajelzést tesz
 - az általa biztosított interfészeket csak egyeztetés után módosítja
 - dokumentálja (kommentezi), illetve teszteli a saját kódját (elkészíti a megfelelő egységteszteket)

Clean Code

Milyen a Clean Code?

- Clean Code-ként hivatkozhatunk arra a programkódra, ami teljesíti az alábbi szubjektív tulajdonságokat
 - Olvasható
 - Karbantartható
 - Tesztelhető
 - Elegáns



Clean Code

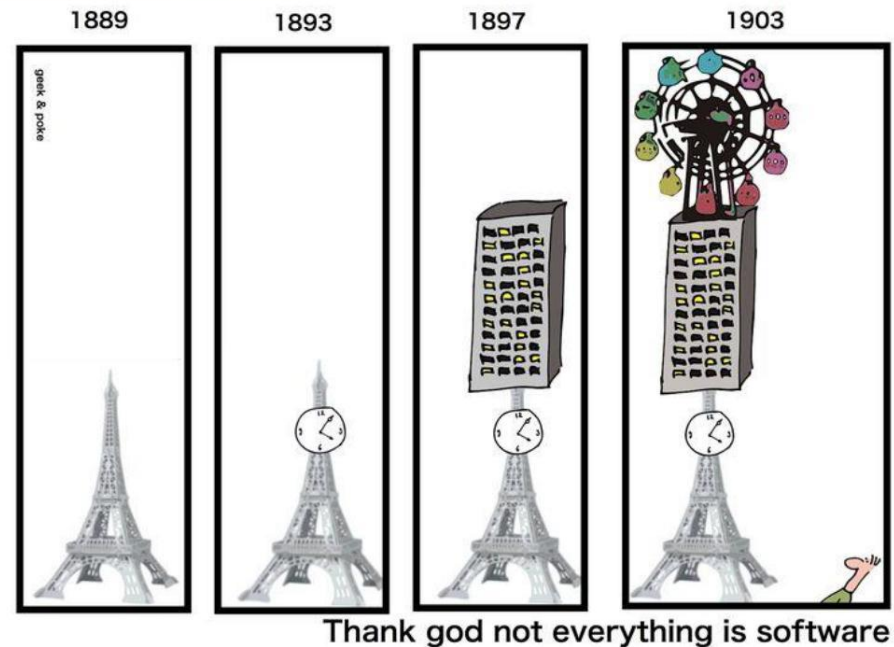
A Clean Code jelentősége

A megírt kód folyton változik

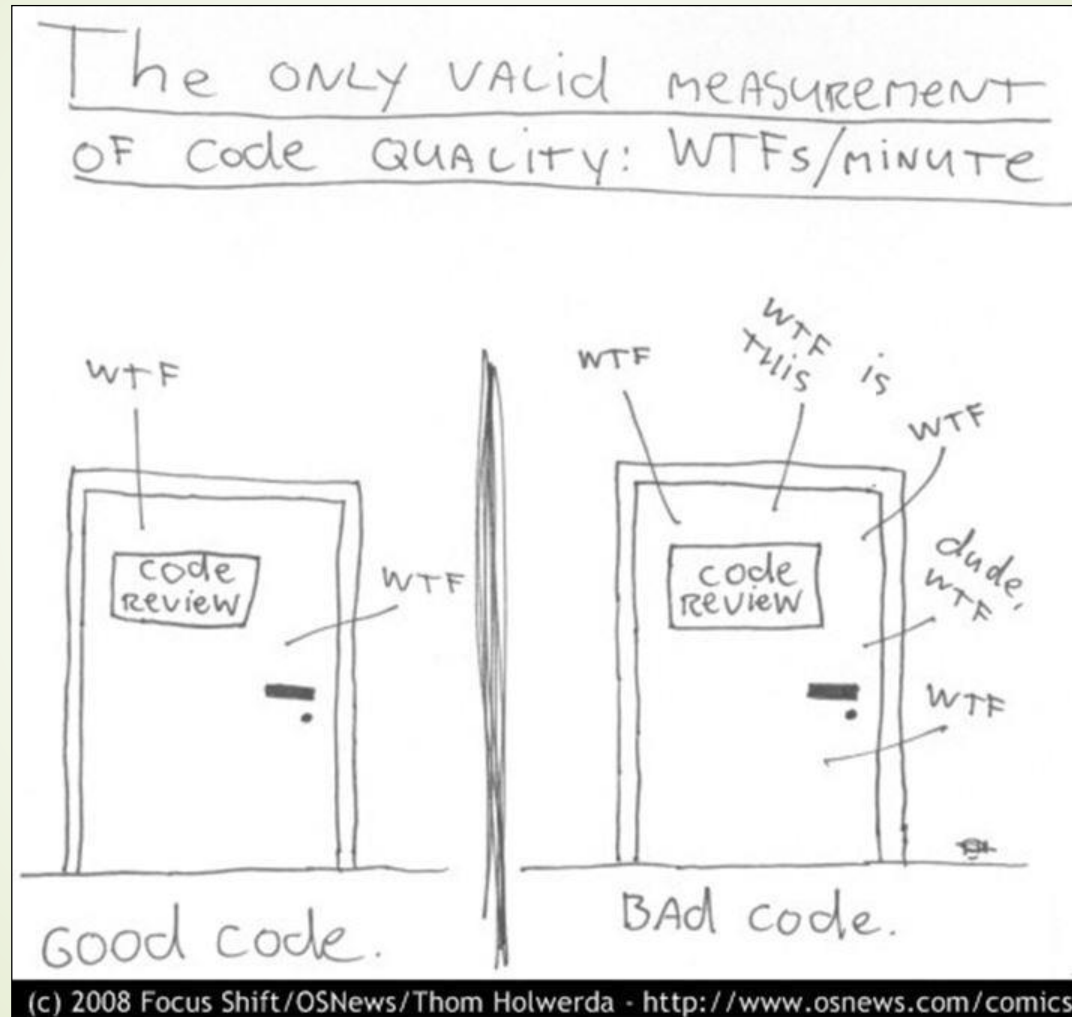
- Hibajavítások
- Követelmény változások
- Új funkciók

Tiszta kód hiányában

- A fejlesztési idő megnő
- Nehezen felderíthető hibák



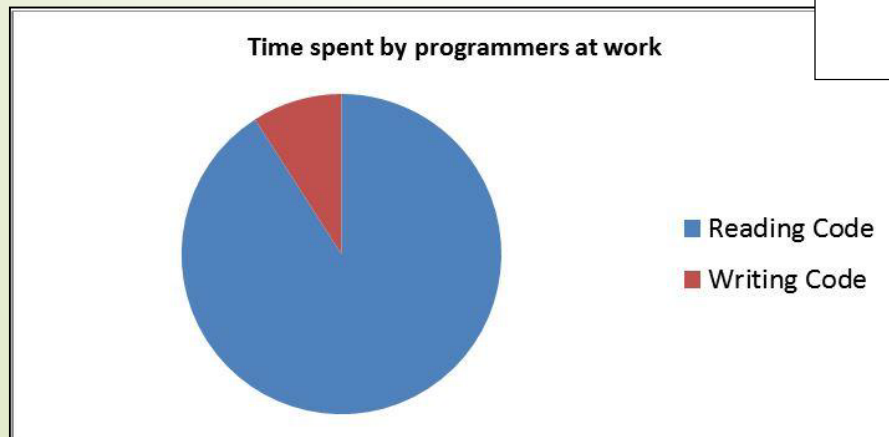
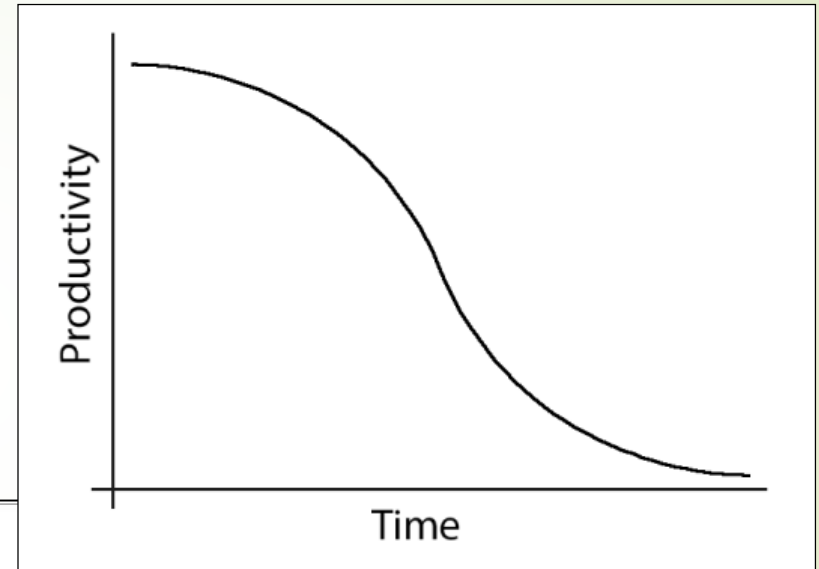
Clean Code



Clean Code

Kód minőség, karbantarthatóság

- A kód minősége közvetlenül hatással van a karbantarthatóságra.



Clean Code

Probléma

- Majd megcsinálom később:
 - Később soha nem jön el
 - A kódot egyszerűbb tisztán tartani, mint később rendbe tenni



Implementáció és verziókövetés

Kódolási stílus

- A *kódolási stílus* (*coding style*) egy szabályhalmaz, amely a forráskód megjelenésére (pl. elnevezés, indentálás, szóközök elhelyezése, ...) ad iránymutatást
 - a kódolási stílus követése javítja a kód értelmezhetőségét, a későbbi karbantartást, és lehetővé teszi a fejlesztők közötti kommunikáció zökkenőmentességét
 - a jó programozási stílus általában szubjektív, nem túl szigorú, de alapvető elemeket definiál
 - a kódolási stílus lehet nyelvi szinten, vállalati szinten, vagy szoftverszinten rögzített
 - a fejlesztőkörnyezetek általában lehetőséget adnak a forrás automatikus formázására

Implementáció és verziókövetés

Kódolási stílus

- Pl.:

```
class Point { // camel case (upper, Pascal case)
private:
    int xCoordinate; // camel case (lower)
    int yCoordinate;

    ...

    double DistanceTo(Point other) const
        // camel case (upper)
    {
        return ...
    } // K&R

    ...

};
```

Implementáció és verziókövetés

Kódolási stílus

- Pl.:

```
class Point {  
private:  
    int x_coordinate; // snake case  
    int y_coordinate;  
  
    ...  
  
    double Distance_To(Point other) const {  
        // Oxford case  
        return ...  
    } // 1TBS  
  
    ...  
};
```

Implementáció és verziókövetés

Kódolási stílus

- Pl.:

```
class Point {
private:
    int m_nXCoord; // hungarian notation
    int m_nYCoord;

    ...

    double distanceTo(Point pOther) const
        // camel case (lower)
    {
        return ...
    } // GNU

    ...
};
```


Implementáció és verziókövetés

Kódolási stílus

- Általános érvényű javaslatok:
 - kódrészletek megfelelő elválasztása (szóköz, sortörés, behúzás, függőleges igazítás)
 - beszédes és konzisztens elnevezések használata (kevesebb kommentezést igényelnek)
 - beégetett tartalmi elemek (számok, szövegek) megnehezítik a karbantartást (*hard coding*), ezért célszerű a kerülése, kiemelése fejlécbe, vagy konfigurációs fájlba (*soft coding*)
- A kódolási konvenció rákényszeríthető a programozóra kódolási stílus ellenőrző eszköz segítségével
 - pl. *C++Test*, *StyleCop*



Clean Code

Elnevezések

„Choosing good names takes time,
but saves more than it takes.”



Clean Code

Elnevezések

Használat módjára utaló elnevezése

`int d; // elapsed time in days`

- Mit fejez ki `d` ? Semmit. Sem az eltelt időre, sem a napokra nem vonatkozik.
- `int elapsedTimeInDays;`

Félrevezető nevek

- `theList` – Not very good
- `ProductList` – A bit better
- `ProductCatalog` – Good

Ugyanazt a nevet ne használjuk különböző célra



Clean Code

Elnevezések

Kiejthető / Kereshető nevek

- Date genymdhms; VAGY Date generationTimestamp;
- Konstansok, előfordulások keresése?
- MAX_ORDER_BY_CUSTOMER vs. 6

Egy szó / koncepció

- Összezavaró ha ugyanarra a dologra több névvel hivatkozunk, pl.:
- fetch, retrieve, get mint ekvivalens metódusok különböző osztályokban.

Kerüljük a név prefixeket

- Project név: „My Project”
- MPOrderService

Implementáció és verziókövetés

Kommentezés

- A kódot a stílusnak megfelelő kommenttel kell ellátni
 - alapvető fontosságú az interfész kommentezése (osztályok, függvények, paraméterek)
 - a megvalósítás kommentezése összetett funkcionalitás esetén hasznos, de megfelelő kódolási stílus esetén nem szükséges
 - tartalmazhat speciális jelöléseket (pl. **TODO**, **FIXME**)
 - a túl kevés, vagy túl sok komment is ártalmas lehet
- A kommentek felhasználhatóak dokumentáció előállítására is (pl. *Doxygen*, *Javadoc*), amennyiben azokat megfelelő séma szerint hozzuk létre

Implementáció és verziókövetés

Kommentezés

- Pl.:

```
// a type representing a 2D point
```

```
class Point {
```

```
...
```

```
// computes the Euclidean distance to another  
// point
```

```
double DistanceTo(Point other) const
```

```
{
```

```
    return sqrt(pow(...) + pow(...));
```

```
}
```

```
...
```

```
};
```

Implementáció és verziókövetés

Kommentezés

- Pl.:

```
// Name: Point
//
// Purpose: This type represents a point in a 2D
// coordinate system.
// Remarks: Is based on double coordinates.
//
// License: LGPL v2.
//
// Author: Roberto Giachetta
// Date: 27/11/2014
// Contact: groberto@inf.elte.hu
...
class Point {
```

Implementáció és verziókövetés

Kommentezés

- Pl.:

```
// Name: distance
// Purpose: This method computes the Euclidean
// distance to another Point instance.
// Remarks: This a query method.
// Parameters: other : another point
// Return value: The distance to the other.
double distance(Point other) const
{
    // uses sqrt and pow functions from math.h
    // formula: ...
    return sqrt(pow(...) + pow(...)) ;
}
...
```


Clean Code

Kommentek

- A kommentek hazudnak: Kód és komment nem él együtt.
- A komment nem javít a rossz kódon, ha nehezen érthető (át kell írni)

Például:

```
// Check to see if the employee is eligible for full benefits  
if ((employee.flags & HOURLY_FLAG) && (employee.age > 65))
```

VAGY

```
if (isEligibleForFullBenefits(employee))
```

Clean Code

Jó kommentek

► Információs kommentek: Javadoc

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

► Szándékot, pontosítást tartalmazó kommentek:

```
// Sample input: Oct 5, 2015 -13:54:15 PDT
```

► Következményre figyelmeztető kommentek:

```
// We do a deep copy of this collection to make  
// sure that updates to one copy do not affect  
// the other
```

► TODO, stb.: kommentek

► JavaDoc?

Clean Code

Rossz kommentek

- Zaj / felesleges kommentek: `// the counter`
`private int counter;`
- Kommentek metódusok helyett
- Tagoló commentek: `// Getters..... //////////////`
- Zárójelek kommentjei: `while(...) {...`
`} // while`
- Kikommentezett kód
- (HTML comment)

Implementáció és verziókövetés

Kódolási stílus

- Általában nyílt és zárt programkódokra más szabályok vonatkoznak
 - *nyílt forráskód* esetén törekedni kell, hogy a kód minél gyorsabban értelmezhető legyen bárki számára
 - követni kell a programozási nyelv tördelési és elnevezési konvencióit
 - a kód megfelelő mennyiségű megjegyzéssel kell ellátni
 - *zárt forráskód* esetén a cél a fejlesztőcsapat minél nagyobb rálátása a kódra
 - törekedni kell, hogy minél nagyobb kódmennyiség legyen egyszerre áttekinthető (kevesebb helyköz és komment)

Implementáció és verziókövetés

Statikus kódelemezés

- A *statikus kódelemzés* (*static code analysis*) lehetővé teszi, hogy a forráskódot még a fordítás előtt előfeldolgozzuk, és a lehetséges hibákat és problémákat előre feltérképezzük
 - a fejlesztőkörnyezet beépített *kódelemző*vel rendelkezhet, amely megadott szabályhalmaz alapján a lehetséges hibaeseteket felfedi
 - a statikus kódelemzés egy része kimondottan a kódolási konvenciók (pl. elnevezések, tagolás, dokumentáltság) ellenőrzését biztosítja
 - a kódra számíthatók *metrikák* (*code metric*), amelyek megadják karbantarthatóságának, összetettségének mértékét (pl. *cyclomatic complexity*, *class coupling*)

Implementáció és verziókövetés

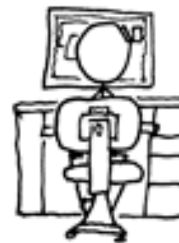
Kód-újratervezés

- A *kód-újratervező* (*refactoring*) eszközök célja, hogy a kód szerkezetét mindig konzisztens módon, a viselkedés befolyásolása nélkül tudjuk módosítani
 - a teljes újratervezést kisebb lépések (*micro-refactorings*) sorozatával érjük el
 - pl. átnevezések, ismétlődő kódok kiemelése, típuscsere, interfész, vagy őssosztály kiemelése, tervezési minta bevezetése
 - a kód nem funkcionális követelményeinek javítására szolgál, általában a karbantarthatóság, illetve a bővíthetőség növelése a cél
 - alkalmas bizonyos rejtett hibák, vagy sebezhetőségek felfedezésére

Clean Code

Függvények, metódusok

- Rövid: maximum egy képernyőre rá kell férnie
- Don't Repeat Yourself (DRY) - copy/paste = bad
- Egyetlen dolgot csinál – Egy metódus, egy absztrakciós szint
Magasabb szintől → részletek
- Blokkoknak egyértelmű be és kilépési pont, ~~break, continue~~



Clean Code

```
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);
    }
}
```

- Új employee type esetén a függvény mérete egyre nő
- Megsérti a Single responsibility Principle-t (több ok miatt is változhat)
- Megsérti az Open Closed Principle-t (új típus hozzáadása után változtatni kell)
- Más függvényekben is fel kell használni ugyanazt a struktúrát:
 - isPayday(Employee e, Date date)
 - deliverPay(Employee e, Money pay)
 - ...
- Switch utasítások elfogadhatóak, ha azok nem ismétlődnek.

Clean Code

Megoldás

```
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 SalariedEmployee(r);  
            default:  
                throw new InvalidEmployeeType(r.type);  
        }  
    }  
}
```



Clean Code

Mellékhatások

- Félrevezetés a függvény feladatával kapcsolatban: a jelzett funkció mellett valami rejtett dolgot is elvégez.

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

Clean Code

Command vs query

- A függvények általában vagy végrehajtanak valamit, vagy válaszolnak valamire, de nem mindkettőt egyszerre:
 - **public boolean** set(String attribute, String value);
 - Beállítja egy mező értékét és jelzi sikeres volt-e. Probléma?
 - **if** (set("username", "tibi"))
 - Mit jelent ez?
 - Megmondja, hogy a username mező értéke már tibi-re van állítva?
 - Megmondja, hogy sikeres volt-e az érték beállítása?

Clean Code

„Always leave the code cleaner than you found it.”

- Felesleges, a változásokat követő leírások
- `i++; // increment i`
- Kikommentezett kódok
- Nem használt függvények / kódok
- Duplikátumok
- Kiválasztó argumentumok (true/false)