Code Smells

What is thta smell? Did you write that code?

Agenda

- day 1
- clean comments
- clean Functions
- clean tests
- clean class

- day 2
- oo programming
- functional programming
- reactive programming
- ddd
- secure code

```
public int x() {
    int q = 0;
    int z = 0;
    for (int kk = 0; kk < 10; kk++) {
        if (l[z] == 10) {
            q += 10 + (l[z + 1] + l[z + 2]);
            z += 1;
        }
        else if (l[z] + l[z + 1] == 10) {
            q += 10 + l[z + 2];
            z += 2;
        } else {
            q += l[z] + l[z + 1];
            z += 2;
        }
    }
    return q;
}
```

Code Smell

CODE SMELLS ARE SYMPTOMS OF POOR DESIGN OR IMPLEMENTATION CHOISES

[Martin Fowler]

Solution to the spaghetti code problem

Tip:
Any fool can write code that a computer can understand.
Good programmers write code that humans can understand.

- Refactoring

```
<table width="109%" height="908" border="0" align="center" cellpadding="0" cellspacing="0">

    <td valign="top" height="113" width="191" bgcolor="#FFFFFF">
      <div align="center"><a href="http://www.pawprintzpetboutique.com/dog-clothes-shirts-s
ics/PS/ps-personalityprincesschloe-th.jpg" alt="dog clothes t-shirt" width="125" height="12
    <td width="191" height="113" valign="top" bordercolor="#7C2953" bgcolor="#FFFFFF">
      <div align="center"><a href="dog-clothes-sweaters.htm"><img src="pics/KW/kw-pp-pnk-th
og sweaters"></a></div></td>
    <td width="191" height="113" valign="top" bordercolor="#7C2953" bgcolor="#FFFFFF">
      <div align="center"><a href="http://www.pawprintzpetboutique.com/dog-clothes-shirts.h
ics/Woof/Summer05/teddy-shirt-sm.jpg" alt="small dog clothes shirts" width="125" height="10
  </tr>
  <td valign="top" height="60" width="191" bgcolor="#FFFFFF"> <div align="center"><b><font
rif"></font><font color="ffcc66" size="2" face="Geneva, Arial, Helvetica, san-serif"><b><a
http://www.pawprintzpetboutique.com/dog-clothes-shirts-sleeves-personality.htm">Personality
Geneva, Arial, Helvetica, san-serif" size="2"></font></b></div></td>
  <td width="191" height="60" valign="top" bordercolor="#7C2953" bgcolor="#FFFFFF">
    <div align="center"><b><font color="ffcc66" face="Geneva, Arial, Helvetica, san-serif"
http://www.pawprintzpetboutique.com/dog-clothes-sweaters.htm">Sweaters</a></font></b></div>
  <td width="191" height="60" valign="top" bordercolor="#7C2953" bgcolor="#FFFFFF">
    <div align="center"><b><font size="2" face="Geneva, Arial, Helvetica, san-serif"><a
http://www.pawprintzpetboutique.com/dog-clothes-shirts.htm">
      Shirts</a></font></b></div></td>
  </tr>
  <td valign="top" height="120" width="191" bgcolor="#FFFFFF"> <div align="center"><a
http://www.pawprintzpetboutique.com/dog-clothes-sweat.htm"><img src="pics/Woof/Summer05/cro
="0" alt="dog clothes sweatshirts"></a></div></td>
  <td valign="top" height="120" width="191" bordercolor="#7C2953" bgcolor="#FFFFFF">
    <div align="center"><a href="http://www.pawprintzpetboutique.com/dog-clothes-pajamas.ht
ics/Woof/Summer05/heart-sm.jpg" alt="dog clothes pajamas" width="125" height="105" border="
  <td valign="top" height="120" width="191" bordercolor="#7C2953" bgcolor="#FFFFFF">
    <div align="center"><a href="http://www.pawprintzpetboutique.com/dog-clothes-coats.htm"
ics/PS/apparel/psblingjacketharley-sm.jpg" width="103" height="104" border="0" alt="dog clo
  </tr>
  <td valign="top" height="60" width="191" bgcolor="#FFFFFF"> <div align="center"><font col
```

Boy Scout Rule

Leave your code better than you found it.

Problem Accumulation

Story

- Area was pristine clean for many days , one fine day a lazy guy threw garbage at the place just one bag , next one week place was filled with garbage , reason other people saw garbage and instead of picking it up they threw their own.

Conclusions

- Fix problems immediately
- Continuous sanitization
- Prevent the impression of hopelessness
  - Hopelessness leads to less ambitious work, which leads to more bugs, which leads to irresponsibility, which creates more hopelessness

```
public class SetScorer
{
    private int[] games = {0, 0};

    public void gameWon(int i) {
        games[i-1]++;
    }

    public String getSetScore() {
        if (games[0] < 6 && games[1] < 6) {
            if (games[0] > games[1]) {
                return "Player1 leads " + games[0] + " - " + games[1];
            } else if (games[1] > games[0]) {
                return "Player2 leads " + games[1] + " - " + games[0];
            } else {
                return "Set is tied at " + games[1];
            }
        }
        if (games[0] == 6 && games[1] < 5) {
            return "Player1 wins the set " + games[0] + " - " +
                  games[1];
        }
        if (games[1] == 6 && games[0] < 5) {
            return "Player2 wins the set " + games[1] + " - " +
                  games[0];
        }
        if (games[0] == 6 && games[1] == 5) {
            return "Player1 leads 6 - 5";
        }
        if (games[1] == 6 && games[0] == 5) {
            return "Player2 leads 6 - 5";
        }
        if (games[0] == 6 && games[1] == 6) {
            return "Set is tied at 6 games";
        }
        if (games[0] == 7) {
            return "Player1 wins the set 7 - " + games[1];
        }
        return "Player2 wins the set 7 - " + games[0];
    }
}
```

```
public class SetScorer {
      private int[] gamesWon = {0, 0};

      public void gameWon(int player) {
            gamesWon[player-1]++;
      }

      public String getSetScore() {
            int leader = gamesWon[0] > gamesWon[1] ? 1 : 2;
            int leadersGames = gamesWon[leader -  1];
            int opponentsGames = gamesWon[leader == 1 ? 1 : 0];
            String setScoreMessage = null;
            if ((gamesWon[0] < 6 && gamesWon[1] < 6)
                  || (leadersGames == 6 && opponentsGames == 5)) {
                      setScoreMessage = "Player" + leader + " leads " +
                        leadersGames + " - " + opponentsGames;
            } else if (gamesWon[0] == gamesWon[1]) {
                      setScoreMessage = "Set is tied at " +
                              leadersGames;
            } else if ((leadersGames - opponentsGames >= 2)
                  || (leadersGames == 7)) {
                  setScoreMessage = "Player" + leader +
                        " wins the set " + leadersGames + " - " +
                        opponentsGames;
            }
            return setScoreMessage;
      }
}
```

Comments

```
InputStreamResponse response = new InputStreamResponse();
response.setBody(formatter.getResultStream(), formatter.getByteCount());
// InputStream resultsStream = formatter.getResultStream();
// StreamReader reader = new StreamReader(resultsStream);
// response.setContent(reader.read(formatter.getByteCount()));
```

Ruthlessly delete code that isn't being used.

```
/*

public void registerItem(Item item) {
      if (item != null) {
            ItemRegistry registry = peristentStore.getItemRegistry();
            if (registry != null) {
                  Item existing = registry.getItem(item.getID());
                  if (existing.getBillingPeriod().hasRetailOwner()) {
                        existing.register(item);
                  }
            }
      }
}

….

*/
```

It is inappropriate for a comment to hold information better held in a as source code control system, issue tracking system, or any other record-keeping system.

```
/**
* @dateModified 12/12/1947
* @modifiedBy RamgopalVerma
* @modifiedReason because sky is so high
* /
public class Patient
{
}
```

Avoid *Obsolete Comment*

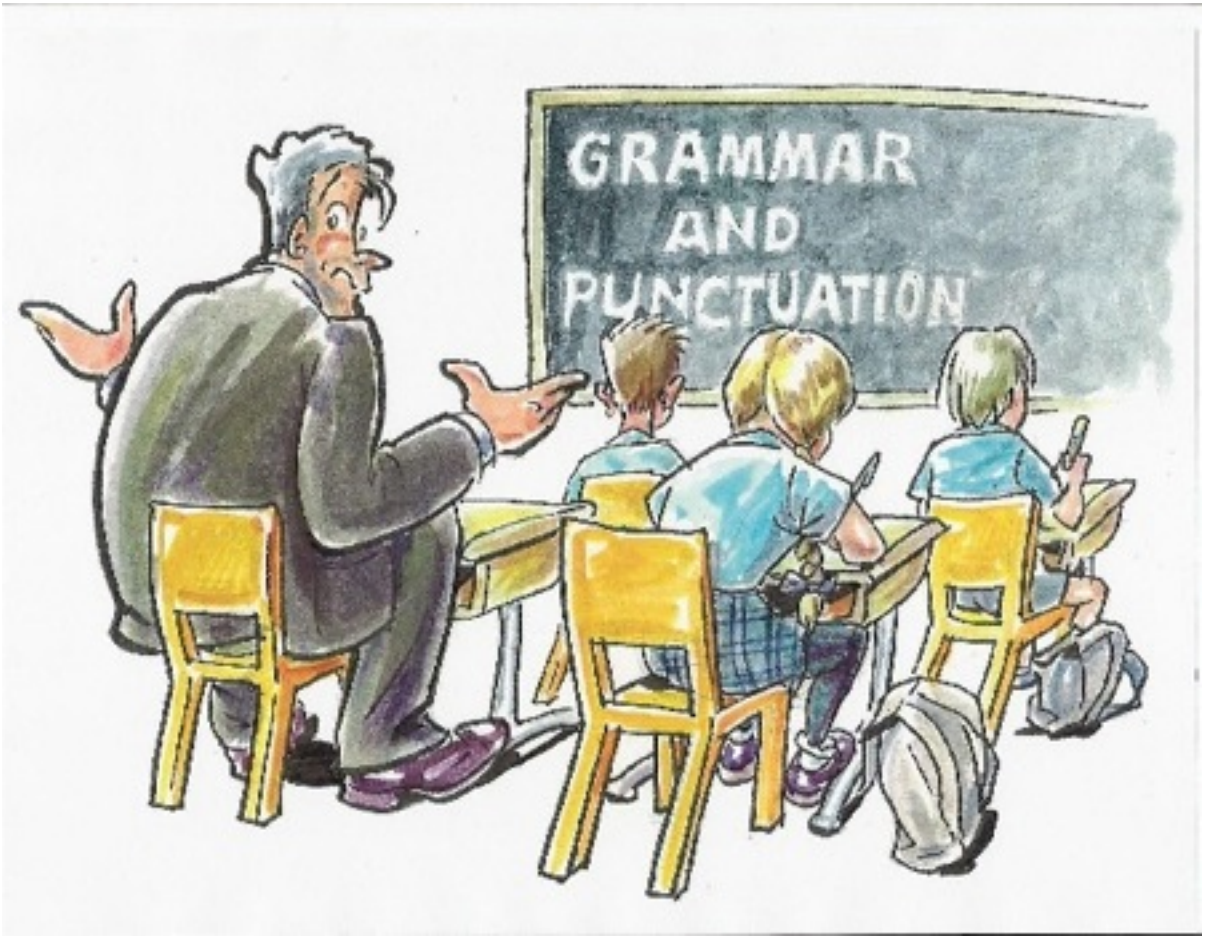
It is best not to write a comment that will become obsolete.

They become floating islands of irrelevance and misdirection in the code.

Dead Function

Methods that are never called should be discarded.

```
patientCount++; // add patient hospital visiting
```



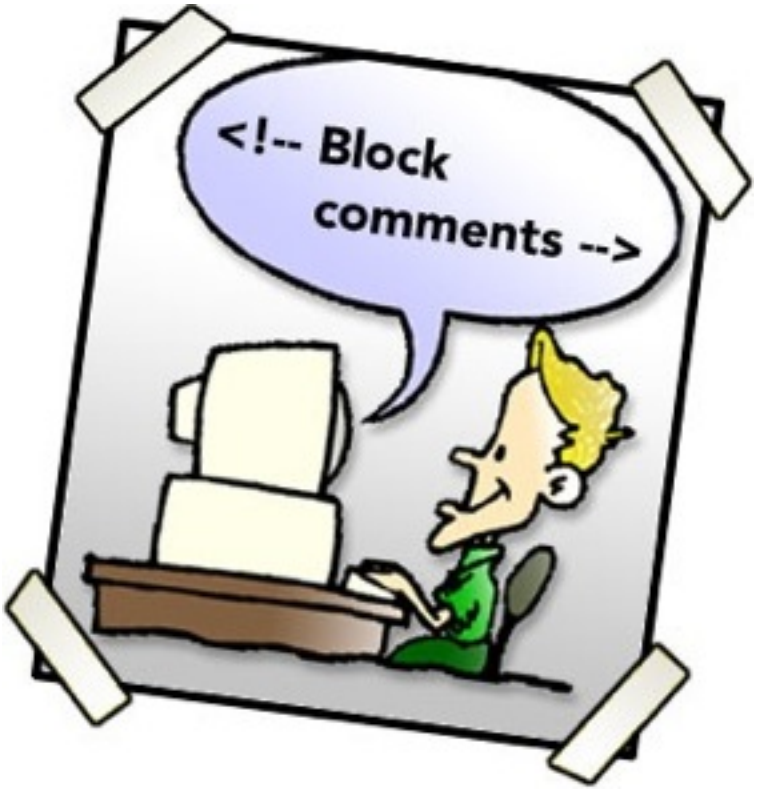
A comment worth writing is worth writing well.

```
i++; // increment I
```

Comments are Failures

they compensate for our inability to express in code

```
/**
* Returns the day of the month.
*
* @return the day of the month.
*/
public int getDayOfMonth() {
   return dayOfMonth;
}
```



Comments should say things that the code cannot say for itself.

```
//Add Books associated with the Category

public void add(Category category)
{
        bookMap.put(category.getCategoryId(),
                        category.getBooks());
}
```

Explain Yourself in Code

```
public void addBooksFromCategory(Category category)
{
  bookMap.put(category.getCategoryId(), category.getBooks());
}
```

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

Explain Yourself in Code

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

```
double getExpenseLimit() {
  // should have either expense limit or a primary project
  return (_expenseLimit != NULL_EXPENSE) ?
    _expenseLimit:
    _primaryProject.getMemberExpenseLimit();
}
```

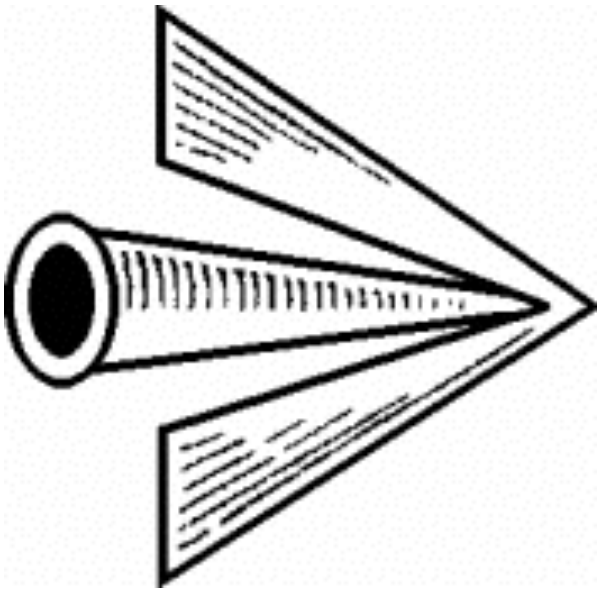
Introduce Assertion

```
double getExpenseLimit() {
  Assert.isTrue (_expenseLimit != NULL_EXPENSE || _primaryProject != null);
  return (_expenseLimit != NULL_EXPENSE) ?
    _expenseLimit:
    _primaryProject.getMemberExpenseLimit();
}
```

Error

```
public class DeviceController {
      ...
      public void sendShutDown() {
            DeviceHandle handle = getHandle(DEV1);
            // Check the state of the device
            if (handle != DeviceHandle.INVALID) {
                  // Save the device status to the record field
                  retrieveDeviceRecord(handle);
                  // If not suspended, shut down
                  if (record.getStatus() != DEVICE_SUSPENDED) {
                        pauseDevice(handle);
                        clearDeviceWorkQueue(handle);
                        closeDevice(handle);
                  } else {
                        logger.log("Device suspended. Unable to shut down");
                  }
            } else {
                  logger.log("Invalid handle for: " + DEV1.toString());
            }
      }
```

Arrow Code



```
if
  if
    if
      if
        do something
      endif
    endif
  endif
endif
```

Use Exceptions Rather Than Return Codes

```
public class DeviceController {
      ...
      public void sendShutDown() {
            try {
                  tryToShutDown();
            } catch (DeviceShutDownError e) {
                  logger.log(e);
            }
      }
      private void tryToShutDown() throws DeviceShutDownError {
            DeviceHandle handle = getHandle(DEV1);
            DeviceRecord record = retrieveDeviceRecord(handle);
            pauseDevice(handle);
            clearDeviceWorkQueue(handle);
            closeDevice(handle);
      }
      private DeviceHandle getHandle(DeviceID id) {
            ...
            throw new DeviceShutDownError("Invalid handle for: " + id.toString());
            ...
      }
      ...
}
```

```
void DoJob()
{
        Domain domain = new Domain();
        bool res = domain.fun();
        if(res == true){
                res = domain.fun2(100);
                if(res == true){
                        Repostory rep = new Repository();
                        Emp emp = rep.get(1);
                        if(emp != null){
                                ...
                        }else{
                                ...
                        }
                }else{
                        ...
                }
        }else{
                ...
        }
}
```

```
void DoJob()
{
        Domain domain = new Domain();
        domain.fun();
        domain.fun2(100);
        Repostory rep = new Repository();
        Emp emp = rep.get(1);
        ...
}


        try{
                DoJob();
        }catch(…){
                ….
        }
```

```
int withdraw(int amount) {
  if (amount > _balance)
    return -1;
  else {
    _balance -= amount;
    return 0;
  }
}
```

Replace Error Code with Exception

```
void withdraw(int amount) throws BalanceException
{
  if (amount > _balance)
        throw new BalanceException();

  _balance -= amount;
}
```

```
double getValueForPeriod (int periodNumber) {
  try {
        return _values[periodNumber];
  } catch (ArrayIndexOutOfBoundsException e) {
        return 0;
  }
}
```

Replace Exception with Test

```
double getValueForPeriod (int periodNumber) {
        if (periodNumber >= _values.length)
                return 0;

        return _values[periodNumber];
}
```

```
private static List readLines(String fileName)
{
    String line;
    ArrayList list= new ArrayList();

    try
   {
      BufferedReader in = new BufferedReader(new FileReader(fileName));
      while ((line = in.readLine()) != null)
        list.add(line);
      in.close();
    }
    catch (Exception e)
    {
      System.out.println(e);
      return null;
    }
    return list;
}
```

Always know why you are catching an exception

```
private static ArrayList readLines(String fileName) throws IOException
 {
   String line;
   ArrayList file = new ArrayList();

   using(BufferedReader in = new BufferedReader(new FileReader(fileName)))
   {

   while ((line = in.readLine()) != null)
     file.add(line);

   }
   return file;
 }
```

```
public void create(Map<String,Object> results) throws Exception {
   //validate condition
   if(condition) {
        results.set("ERROR_CODE","CONDITION VIOLATED");
   }
}

Map<String,Object> results = new HashMap<>();
create(results);
if(results.containsKey("ERROR_CODE")) {
   ...
}
```

Mixing error handling with output
Use Exceptions

```
public void create(Map<String,Object> results) throws Exception {
   //validate condition
   if(condition) {
        throw new ConditionViolatedException();
   }
}

try {
   Map<String,Object> results = new HashMap<>();
   create(results);
}
catch(ConditionViolatedException exp) {

}
```

```
String line;
ArrayList file = new ArrayList();

try
{
  BufferedReader in = new BufferedReader(new FileReader(fileName));
  while ((line = in.readLine()) != null)
  {
    file.add(line);
  }
  in.close();
}
catch (Exception e)
{
}
```

```
//method 1
catch(SQLException exp) {
   log.fatal("Exception occurred:", exp);
   throw exp;
}

//method 2
try {
   //call method 1
}
catch(Exception exp) {
   log.fatal("Exception occurred:", exp);
}
```

Log exception stack trace only once

```
//method 1
catch(SQLException exp) {
   log.fatal("Exception occurred:", exp);
   throw exp;
}

//method 2
try {
   //call method 1
}
catch(Exception exp) {

}
```

```
catch(InterruptedException exp) {
   AppException exp = new AppException();
   exp.addError(Constants.ERROR_CODE, "EXP005");
   exp.addError(Constants.ERROR_MESG, exp.getMessage());
   throw exp;
}
catch(SQLException exp) {
   AppException exp = new AppException();
   exp.addError(Constants.ERROR_CODE, "EXP011");
   exp.addError(Constants.ERROR_MESG, exp.getMessage());
   throw exp;
}
catch(Exception exp) {
   AppException exp = new AppException();
   exp.addError(Constants.ERROR_CODE, "EXP003");
   exp.addError(Constants.ERROR_MESG, exp.getMessage());
   throw exp;
}
```

remove duplicate code

```
private AppException createAppException(String errorCode) {
   AppException exp = new AppException();
   exp.addError(Constants.ERROR_CODE, errorCode);
   exp.addError(Constants.ERROR_MESG, exp.getMessage());
   return exp;
}

//code
catch(InterruptedException exp) {
   throw createAppException("EXP005");
}
catch(SQLException exp) {
   throw createAppException("EXP011");
}
catch(Exception exp) {
   throw createAppException("EXP003");
}
```

```
void fun()
{
	try
	{
		…logic 1

		if(cond)
			throw new Exception(…);

		…logic 2

	}
	catch(Exception e)
	{
		…logic 3
	}
}
```

```
try {
      …

} catch (Exception e) {
        LOGGER.error("Exception :", e);
        LOGGER.debug("Exception :" + e.getMessage());
        try {
                con.rollback();
        } catch (SQLException e1) {
                LOGGER.error("Exception :", e1);
                LOGGER.debug("Exception :" + e1.getMessage());
        }
} finally {
        try {
                con.close();
        } catch (SQLException e1) {
                LOGGER.error("Exception :", e1);
                LOGGER.debug("Exception :" + e1.getMessage());
        }
}
```

functions

```
while ((!found) && (pos < (fileContent.Length - 6)))
{
    byteData = new byte[6];
    Array.Copy(fileContent, pos, byteData, 0, 6);
    pos = pos + 6;
    str_byteData = enc.GetString(byteData);
    if (str_byteData.Contains("s"))
    {
        posE_byteData = str_byteData.IndexOf("s");
        pos = pos + (posE_byteData - 6);
        Array.Copy(fileContent, pos, byteData, 0, 6);
        pos = pos + 6;
        if (byteData[0] == 0x73)    // 's'
        {
            if (byteData[1] == 0x74)    // 't'
            {
                if (byteData[2] == 0x72)    // 'r'
                {
                    if (byteData[3] == 0x65)    // 'e'
                    {
                        if (byteData[4] == 0x61)    // 'a'
                        {
                            if (byteData[5] == 0x6D)    //
                            {
                                found = true;
                                break;
                            }
                            else
                            {
                                if (byteData[5] == 0x73)
                                {
                                    pos = pos - 1;
```

Nesting Levels

{if < 4 , switch,while,do, for <3}

```
double potentialEnergy(double mass, double height) {
  return mass * height * 9.81;
}
```

Magic Numbers and Strings

Explicit is better than implicit. - Tim Peters, The Zen of Python

Replace Magic Number with Symbolic Constant

```
double potentialEnergy(double mass, double height) {
  return mass * GRAVITATIONAL_CONSTANT * height;
}
static final double GRAVITATIONAL_CONSTANT = 9.81;
```

Create a class for constants

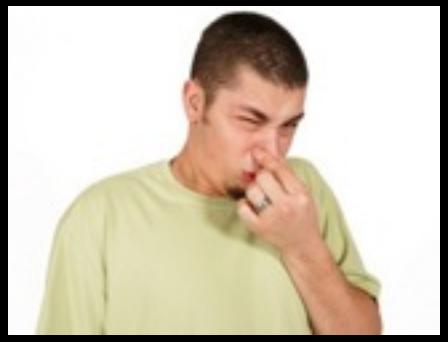
```
public class Constants {
   public static final String Y = "Y";
}

if(validate.equals(Constants.Y)) {

}
```

- DO NOT create a dreadful 4000-lines Constants class !

Define constants in properties file

```
Constants.properties
VALIDATION_FLAG=Y

public class Constants {
   private ResourceBundle bundle =
   ResourceBundle.getBundle("Constants");
   public static final String VALIDATION_FLAG =
        bundle.getProperty("VALIDATION_FLAG");
}

//usage in code
if(validate.equals(Constants.VALIDATION_FLAG)) {

}
```

```
class Program
{
    static void Main(string[] args){
        DoSomething();
        DoSomethingAgain();
        DoSomethingMore();
        DoSomethingExtraordinary();
        Console.ReadLine();
    }
    private static void DoSomething(){
        string address = "Stockholm, Sweden";
        string format = "{0} is {1}, lives in {2}, age {3}";
        Console.WriteLine(format, "Nils", "a good friend", address, 30);
    }
    private static void DoSomethingAgain(){
        string address = "Stockholm, Sweden";
        string format = "{0} is {1}, lives in {2}, age {3}";
        Console.WriteLine(format, "Christian", "a neighbour", address, 54);
    }
    private static void DoSomethingMore() {
        string address = "Stockholm, Sweden";
        string format = "{0} is {1}, lives in {2}, age {3}";
        Console.WriteLine(format, "Eva", "my daughter", address, 4);
    }
    private static void DoSomethingExtraordinary(){
        string address = "Stockholm, Sweden";
        string format = "{0} is {1}, lives in {2}, age {3}";
        Console.WriteLine(format, "Lilly", "my daughter's best friend", address, 4);
    }
}
```



DON'T REPEAT YOURSELF

Repetition is the root of all software evil

Disable the paste function on every developer machine

```
class Program
   {
      private static string address = Constants.Address;
      private static string format = Constants.StandardFormat;

      static void Main(string[] args)
      {
         DoSomething();
         DoSomethingAgain();
         DoSomethingMore();
         DoSomethingExtraordinary();
         Console.ReadLine();
      }

      private static void DoSomething()
      {
         Console.WriteLine(format, "Nils", "a good friend", address, 30);
      }

      private static void DoSomethingAgain()
      {
         Console.WriteLine(format, "Christian", "a neighbour", address, 54);
      }

      private static void DoSomethingMore()
      {
         Console.WriteLine(format, "Eva", "my daughter", address, 4);
      }

      private static void DoSomethingExtraordinary()
      {
         Console.WriteLine(format, "Lilly", "my daughter's best friend", address, 4);
      }
   }
```

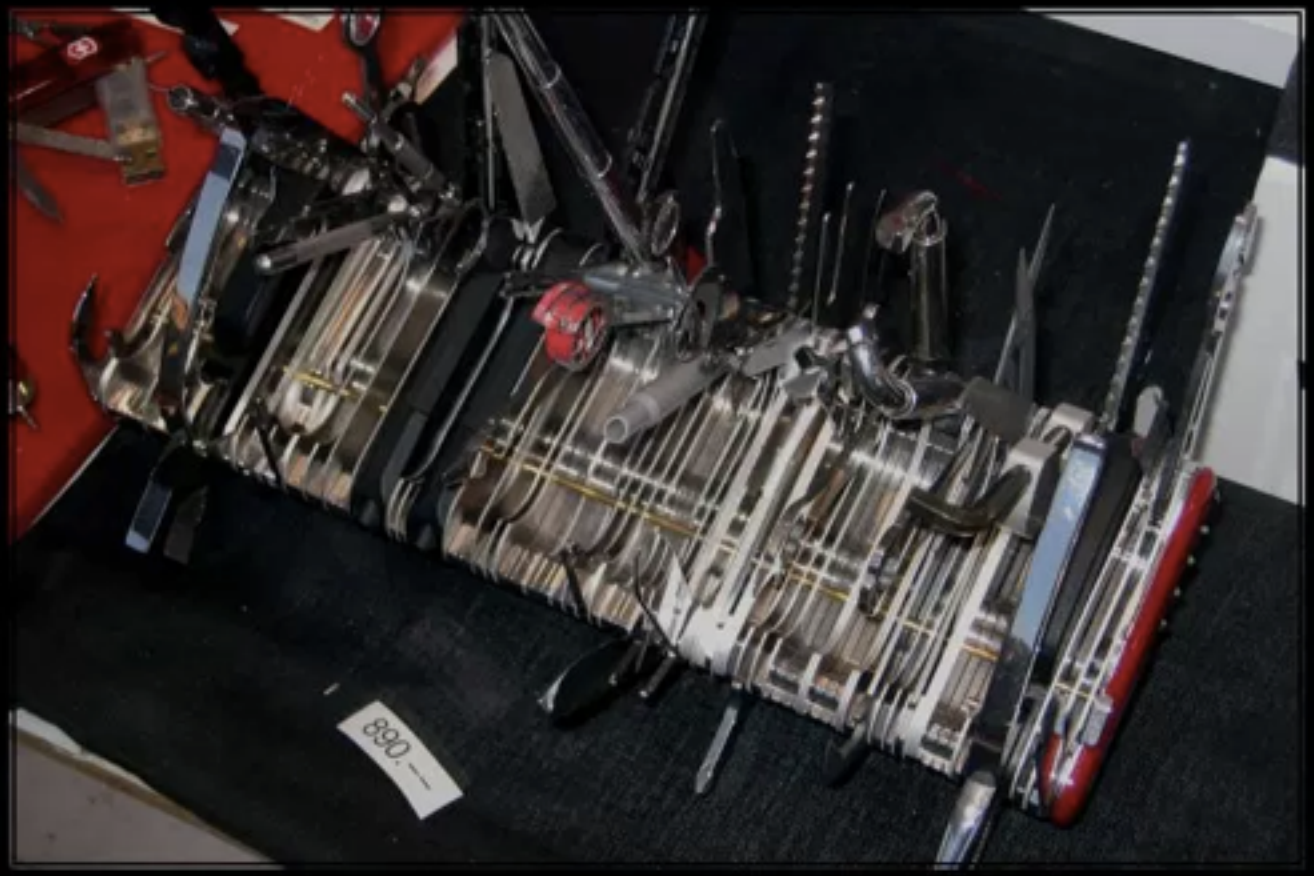
```
public class Employee
{
    public string Name { get; set; }
    public int Age { get; set; }
    public string Department { get; set; }
}

private static IEnumerable<Employee> GetEmployees()
{
    return new List<Employee>()
    {
        new Employee(){Age = 30, Department="IT", Name="John"}
        , new Employee(){Age = 34, Department="Marketing", Name="Jane"}
        , new Employee(){Age = 28, Department="Security", Name="Karen"}
        , new Employee(){Age = 40, Department="Management", Name="Dave"}
    };
}
```

“Perfection (in enterprise development) is achieved not when there is nothing more to add, but rather when there is nothing more to take away.”

--Antoine de Saint-Exupéry

```
        FormView1.FindControl("noteRow").Visible = true;
        FormView1.FindControl("RequiredFieldValidator4").Visible = false;
        // TODO: Fix this - What if Action is 2nd parameter in URL
        if (Request.Path.Contains(@"ClientProfile.aspx?Action=Add"))
        {
            FormView1.FindControl("thLoginID").Visible = true;
            FormView1.FindControl("tdLoginID").Visible = true;
            FormView1.FindControl("txtLoginID").Visible = true;
        }

    }
    else
    {
        FormView1.FindControl("noteRow").Visible = false;
    }


if (HttpContext.Current.User.IsInRole("Administrator"))
{

    if (!Request.Path.Contains(@"/AdministratorProfile.aspx"))
    {
        if (!Request.Path.Contains(@"/Profile.aspx"))
        {
            FormView1.FindControl("txtLoginID").Visible = false;
            FormView1.FindControl("lblLoginID").Visible = false;
            -- Snip snip snip - more controls are hidden or shown -- --

            if (Request.Path.Contains(@"/UserProfile.aspx") && HttpContext.Current.User.IsInRole("User") && (Request.QueryString["Action"] == null ||
            {

                FormView1.FindControl("thLoginID").Visible = true;
                FormView1.FindControl("lblLoginID").Visible = true;
                -- Snip snip snip - more controls are hidden or shown -- --

            }
            else
            {
                FormView1.FindControl("thLoginID").Visible = false;
                FormView1.FindControl("lblLoginID").Visible = false;
                -- Snip snip snip - more controls are hidden or shown -- --


            }

        }
        else
        {
            FormView1.FindControl("thDateRegistered").Visible = false;
            FormView1.FindControl("tdDateRegistered").Visible = false;
            -- Snip snip snip - more controls are hidden or shown -- --

        }
    }
    else if (Request.Path.Contains(@"/ReaderProfile.aspx"))
    {
        FormView1.FindControl("tdAdviser").Visible = false;
        FormView1.FindControl("thAdviser").Visible = false;
        -- Snip snip snip - more controls are hidden or shown -- --

    }
}
else if (HttpContext.Current.User.IsInRole("Administrator"))
```

Long Method



Single Responsibility Principle

Yes, it's all there. No, it's not a good idea.

Functions Should be Small !!

- How small ?
  - **4 *lines*** is best; 5 is ok; 6 is already too much
  - By any means, keep your functions smaller than a page of your IDE !

- Why so small ?!

➔To be sure that they do only 1 THING

  ➔In a few lines, you can't do much

  ➔Its name can then tell you everything

```
Public void doTheDomesticThings()
{
        takeOutTheTrash();
        walkTheDog();
        for(Dish dish : dirtyDishStack)
        {
                sink.washDish(dish);
                teaTowel.dryDish(dish);
        }
}
```

Don’t mix levels of abstraction

```
Public void doTheDomesticThings()
{
        takeOutTheTrash();
        walkTheDog();
        doTheDishes();
}
```

```
double getPrice()
{
        int basePrice = _quantity * _itemPrice;

        double discountFactor;
        if(basePrice > 1000)
                discountFactor = 0.95;
        else
                discountFactor = 0.98;

        return basePrice * discountFactor;
}
```

Replace Temp with Query

```
double getPrice()
{
        return basePrice() * discountFactor();
}
private double discountFactor()
{
        if(basePrice > 1000)
                return 0.95;
        else
                return  0.98;
}
private int basePrice()
{
        return _quantity * _itemPrice;
}
```

```
File CreateFile(string name, boolean isTemp)
{
   //implementation
}


myfile = CreateFile("foo.txt", true);
```

Function Signature

- Max 2 parameters
  - For ≥ 3 parameters, it's hard to read: their order
  - ➔ group them in a **Parameter Object**/**DTO**
- Avoid output parameters
  - The caller may be surprised that an arg
- No flag parameters (boolean,
  - It shouts that the function does more than one thing !
  - ➔ Split it into smaller functions

```
myBadMethod("John","Michael"
,
      "Paris","St.
Albergue");
```

```
params.setStreet("Paris")
;…
myBadMethod(params);
```

```
checkCustomer(customer,
order);
```

?!

```
);
```

```
removeOrders(customer, false,
true);
```

Reveal Intend

```
File CreatePermenantFile(String name);
File CreateTempFile(String name);
```

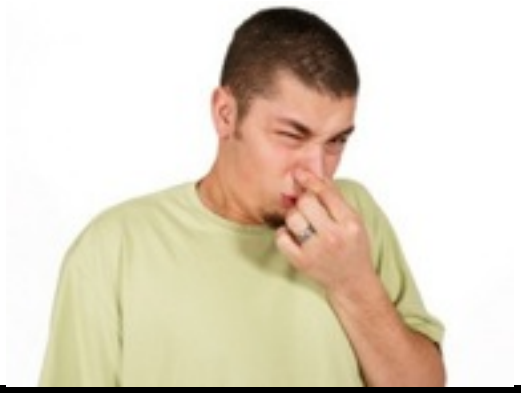
Reveal Intend

```
myFile = CreateFile("foo.txt", FileType.Temp);
```

Reveal Intend

```
isTemp = true;
CreateFile("foo.bar", isTemp);
```

If you have to call an existing API that uses this style,
you should introduce a local variable for clarity.



```
render(true)

void render(boolean isSuite)
{
        …
}
```

Flag Arguments are Ugly

```
void renderForSuite()
{
    ….
}

void  renderForSingleTest()
{
    …..
}
```

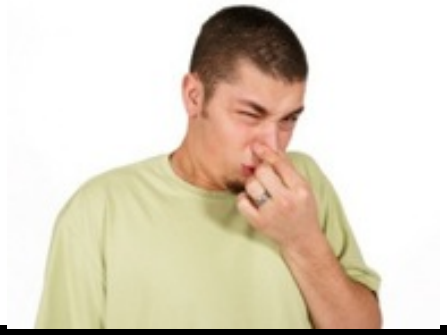
Use descriptive variable names

```
private static List readLines(String fileName) throws IOException
{
   String line;
   List lines = new ArrayList();

   BufferedReader in = new BufferedReader(new FileReader(fileName));

   while ((line = in.readLine()) != null)
     lines.add(line);

   in.close();

   return lines;
  }
```

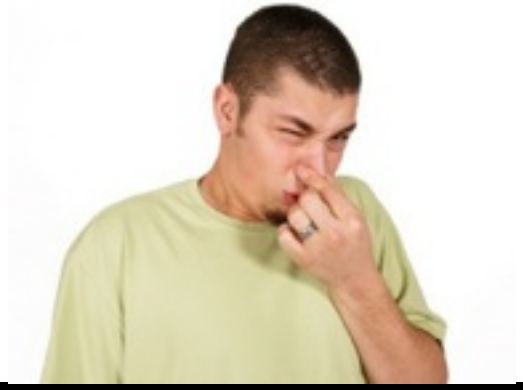
```
void bad_handle_data(char *data, size_t length)
{
        if (check_CRC(data, length) == OK)
        {
                /* * 30 * lines * of * data * handling */
        }
        else
        {
                printf("Error: CRC check failed\n");
        }
}
```

Test Exception instead of normal code

```
void good_handle_data(char *data, size_t length)
{
        if (check_CRC(data, length) != OK)
        {
                printf("Error: CRC check failed\n");
                return;
        }

        /* * 30 * lines * of * data * handling */

}
```

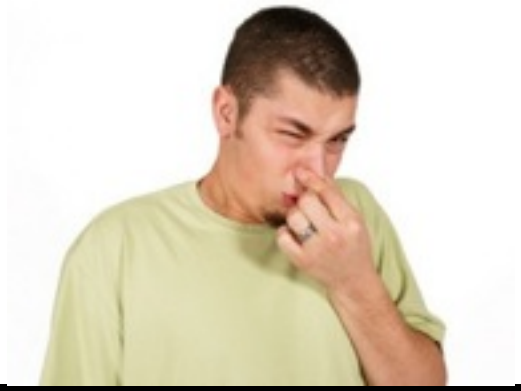


```
public void registerItem(Item item) {
    if (item != null) {
        ItemRegistry registry =
        peristentStore.getItemRegistry();
        if (registry != null) {
            Item existing = registry.getItem(item.getID());
            if (existing.getBillingPeriod().hasRetailOwner()) {
                existing.register(item);
            }
        }
    }
}
```



```
Customer c = findCustomer(...);
...

   if (customer == null) {
          name = “occupant”
   } else {
          name = customer.getName()
   }

if (customer == null) {
…
}
```

```
public void registerItem(Item item) {
      if (item != null) {
            ItemRegistry registry = peristentStore.getItemRegistry();
            if (registry != null && registry.Count > 0) {
                  Item existing = registry.getItem(item.getID());
                  if (existing !=null && existing.getBillingPeriod().hasRetailOwner()) {
                        existing.register(item);
                  }
                  if (existing !=null && !existing.getBillingPeriod().hasRetailOwner()) {
                        existing.unregister(item);
                  }
                  if (existing !=null) {
                        existing.remove(item);
                  }
            }
      }
}
```



```
List<Employee> employees = getEmployees();
if (employees != null) {
    for(Employee e : employees) {
        totalPay += e.getPay();
    }
}

....
```

Don’t Return Null

If we change getEmployee so that it returns an empty list

```
List<Employee> employees = getEmployees();
for(Employee e : employees) {
        totalPay += e.getPay();
}
```

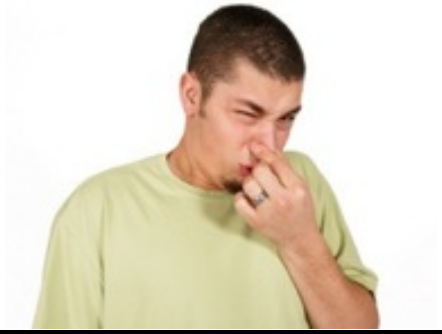
Fortunately, Java has Collections.emptyList(), and it returns a predefined immutable

```
public List<Employee> getEmployees() {
if( .. there are no employees .. )
        return Collections.emptyList();
}
```

Function Signature (2)

- No nullable parameters
  - It's like a `boolean` ⇨ split that function in 2: one for null, one for not-null
  - Thoroughly check parameters at boundaries (=**defensive programming**)
- Don't return null
  - Null is the worst mistake in IT [dzone]

  ➔ Consider Optional<> / throw exception / **Null Object Pattern**
- Unchecked Exceptions won !
  - No more `try`/`throws`/empty `catch`(); no return status codes
  - Runtime Exceptions don't annoy the caller
    - Can be handled uniformly in a transparent layer at the entry points
    - Can hold enum error codes for (i18n) user messages
    - Define custom exception sub-types for recoverable errors (selective catch)



```
public WebUser getCurrentUser(){
	if (_currentUser == null) {
	    Object obj  = HttpContext.Current.Session["__currentUser"];
	    if (obj != null) {
	        _currentUser = (WebUser)obj;
	        return _currentUser;
	    }
	    SecurityHelper secHelper = new SecurityHelper();
	    WebUserRepository rep = new WebUserRepository();
	    if (secHelper.TrackingGuid != Guid.Empty){
	        _currentUser = rep.GetWebUserByTrackingGuid(secHelper.TrackingGuid);
	        if (_currentUser != null)
	       return _currentUser;
	    }
	    WebUserFactory factory = new WebUserFactory();
	    _currentUser = factory.CreateWebUser();
	}
	return _currentUser;
}
```

Avoid multiple returns

```
private WebUser _currentUser;
 public WebUser getCurrentUser()
{
  if (_currentUser == null) _currentUser = GetWebUserFromSession();
     if (_currentUser == null) _currentUser = GetWebUserFromTrackingCookie();
     if (_currentUser == null) _currentUser = CreateNewWebUser();
     return _currentUser;
}
private WebUser GetWebUserFromSession()
{
     Object obj = HttpContext.Current.Session["__currentUser"];
     return obj == null ? null : (WebUser)obj;
}
private WebUser GetWebUserFromTrackingCookie()
{
     SecurityHelper secHelper = new SecurityHelper();
     WebUserRepository rep = new WebUserRepository();
     if (secHelper.TrackingGuid == Guid.Empty)
          return null;
     else
          return rep.GetWebUserByTrackingGuid(secHelper.TrackingGuid);
}
private WebUser CreateNewWebUser()
{
     WebUserFactory factory = new WebUserFactory();
     return factory.CreateWebUser();
}
```

```
double getPayAmount() {
  double result;
  if (_isDead) result = deadAmount();
  else {
    if (_isSeparated) result = separatedAmount();
    else {
      if (_isRetired) result = retiredAmount();
      else result = normalPayAmount();
    };
  }
  return result;
};
```

Replace Nested Conditional with Guard Clauses

```
double getPayAmount() {
  if (_isDead) return deadAmount();
  if (_isSeparated) return separatedAmount();
  if (_isRetired) return retiredAmount();
  return normalPayAmount();
};
```

```
public List<Integer> stringsToInts(
                List<String> strings) {
    if (strings != null) {
        List<Integer> integers = new ArrayList<>();
        for (String s : strings) {
            integers.add(Integer.parseInt(s));
        }
        return integers;
    } else {
        return null;
    }
}
```

use Early returns

```
public List<Integer> stringsToInts2(
                    List<String> strings) {
   if (strings == null) {
      return null;
   }
   List<Integer> integers = new ArrayList<>();
   for (String s : strings) {
      integers.add(Integer.parseInt(s));
   }
   return integers;
}
```

- Reduces the number of indentations

```
if (rowCount > rowIdx) {
      if(drc[rowIdx].Table.Columns.Contains("avalId")) {
      do {
            if (Attributes[attrVal.AttributeClassId] == null) {
            // do stuff
            }
            else {
                  if (!(Attributes[attrVal.AttributeClassId] is ArrayList)) {
                   // do stuff
            }
            else {
                  if (!isChecking) {
                  // do stuff
                  } else
                  {
                  // do stuff
                  } }
                  }
            rowIdx++;
      } while(rowIdx < rowCount && GetIdAsInt32(drc[rowIdx]) == Id);
}
else rowIdx++;
}
return rowIdx;
```

Cyclometic Complexity

```
public void ProcessPages()
{
 while(nextPage !=true)
 {
  if((lineCount<=linesPerPage) && (status != Status.Cancelled)
&& (morePages == true))
  {
   //....
  }
 }
}
```

Cyclometic Complexity

```
public int getValue(int param1)
{
 int value = 0;
 if (param1 == 0)
 {
  value = 4;
 }
 else
 {
  value = 0;
 }
 return value;
}
```

Cyclometic Complexity

CC Value	**Risk**
1-10	Low risk program
11-20	Moderate risk
21-50	High risk
>50	Most complex and highly unstable method

```
private int x;
public int increment_and_return_x()
{
  x = x + 1;
  return x;
}
```

Command / Query Separation

1. A **Query** doesn't have side effects and returns a result

e.g. `getState():State`, `isLoggedIn():boolean`, `findCustomer():Customer`

2. A **Command** changes the system state and returns `void`

e.g. `processOrder(Order)`, `consumeMessage(Message)`, `setName(String)`

- Avoid doing both in the same method

e.g. `processOrder(Order):boolean` (that returns `false` if it failed)

(It forces the caller to check the result right away. Prefer exceptions instead!)

```
private int x;
public int value()
{
  return x;
}
void increment()
{
  x = x + 1;
}
```

```
string result = _storeService.PurchaseItem(buyer, item);
```

Data Structure

```
Map sensors = new HashMap();

…

Sensor s = (Sensor)sensors.get(sensorId );
```

```
Map<int, Sensor> sensors = new HashMap<int, Sensor>();

...

Sensor s = sensors.get(sensorId );
```

Hide boundary (Map)

```
public class Sensors
 {
    private Map sensors = new HashMap();

    public Sensor getById(String id)
    {
        return (Sensor) sensors.get(id);
    }
    //snip
}
```

Code at the boundaries needs clear separation

Encapsulate Collection

```
private ArrayList fieldDetails = new ArrayList();
private HashMap linkedFields = new HashMap();


private void setTransactionParams(Hashtable params) {
    ...
}
```

Use collection framework interfaces, use generics

```
private List<Field> fieldDetails = new ArrayList<>();
private Map<String,FieldProperties> linkedFields = new
   HashMap<>();


private void setTransactionParams(Map<String,Object> params) {
   ...
}
```

```
private void setParameters(HashMap<String,String> parameters) {

}
```

Use interfaces instead of classes

```
private void setParameters(Map<String,String> parameters) {

}
```

```
Outer outer = getObj();

if (outer != null && outer.nested != null && outer.nested.inner != null) {
    System.out.println(outer.nested.inner.foo);
}
```

Removing null check

```
Optional.of(new Outer())
    .map(Outer::getNested)
    .map(Nested::getInner)
    .map(Inner::getFoo)
    .ifPresent(System.out::println);
```

```
int? length = customers?.Length; // null if customers is null
Customer first = customers?[0];  // null if customers is null
int? count = customers?[0]?.Orders?.Count();  // null if customers, the first customer, or Orders is null
```

basics

```
private List getTableColumns(Map tableFieldListMap) {
    Set tables = tableFieldListMap.keySet();
    Iterator iterator = tables.iterator();
    while(iterator.hasNext()) {
        String tableName = (String)iterator.next();
        ...
    }
}
```

Use generics, for..each loop

```
private List<String> getTableColumns(Map<String,Field>
   tableFieldListMap) {
   for(String tableName : tableFieldListMap.keySet()) {
        ...
   }
   ...
}
```

```
private static final String _ALPHABETS = "ALPHABETS";
private static final int _ALPHABETS_NO = 1;

private int getSwitchId(String value) {
        if(value.equals(_ALPHABETS)) {
                return _ALPHABETS_NO;
        }
        ...
}

int switchId = getSwitchId(data);
switch(switchId) {
   case _ALPHABETS_NO:
        ...
}
```

Use enum

```
private enum DataType {
   ALPHABETS,ALPHA_NUMERIC,NUMERIC,SWIFT,ALL,DATE,TIME
}


switch(DataType.valueOf(data)) {
   case ALPHABETS:
        ...
}
```

```
public String getUpdateQuery() {
   StringBuffer updateQuery = new StringBuffer();
   ...
   return updateQuery.toString();
}
```

Use StringBuilder for local variables

```
public String getUpdateQuery() {
   StringBuilder updateQuery = new StringBuilder();
   ...
   return updateQuery.toString();
}
```

```
strName = TextBox1.Text;
iAge = int.parse(TextBox2.Text);
```

Avoid Type Embedded in Name

```
name = TextBox1.Text;
age = int.pasre(TextBox2.Text);
```

Avoid placing types in method names; it's not only redundant, but it forces you to change the name if the type changes.

```
void Clean(int id)
{
        Person object = myCollection.get(id);
        boolean present = myCollection.remove(object);
}
```

Don't preserve return values you don't use

```
void Clean(int id)
{
        Person object = myCollection.get(id);
        myCollection.remove(object);
}
```

```
public class ServiceLocator {
   Map<String,Object> cache;

   public ServiceLocator() {
        cache = Collections.synchronizedMap(
                        new HashMap<String, Object>());
   }
}
```

ConcurrentHashMap has better performance than Collections.synchronizedMap

```
public class ServiceLocator {
   ConcurrentMap<String,Object> cache;

   public ServiceLocator() {
        cache = new ConcurrentHashMap<String, Object>());
   }
}
```

```
Map<String,String> parameters = new Hashtable<>();
```

ConcurrentHashMap has better performance than Hashtable

```
Map<String,String> parameters = new ConcurrentHashMap<>();
```

```
if(condition)
  do something
else
  do something else
```

Use curly braces for if..else blocks

```
if(condition) {
  //do something
}
else {
  //do something else
}
```

```
public String getColumns(List<Field> fields) {
  String column = "";
  StringBuilder columns = new StringBuilder();
  for(Field field : fields) {
     column = field.getColumn();
     columns.append(column);
     columns.append(",");
  }
  return columns.toString();
}
```

Declare variable only where required

```
public String getColumns(List<Field> fields) {
  StringBuilder columns = new StringBuilder();
  for(Field field : fields) {
     String column = field.getColumn();
     columns.append(column);
     columns.append(",");
  }
  return columns.toString();
}
```