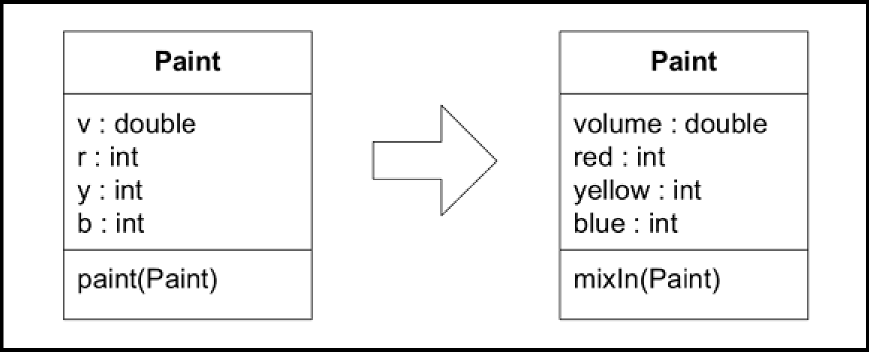
Rule: Don't abbreviate

- Code should be self documenting.

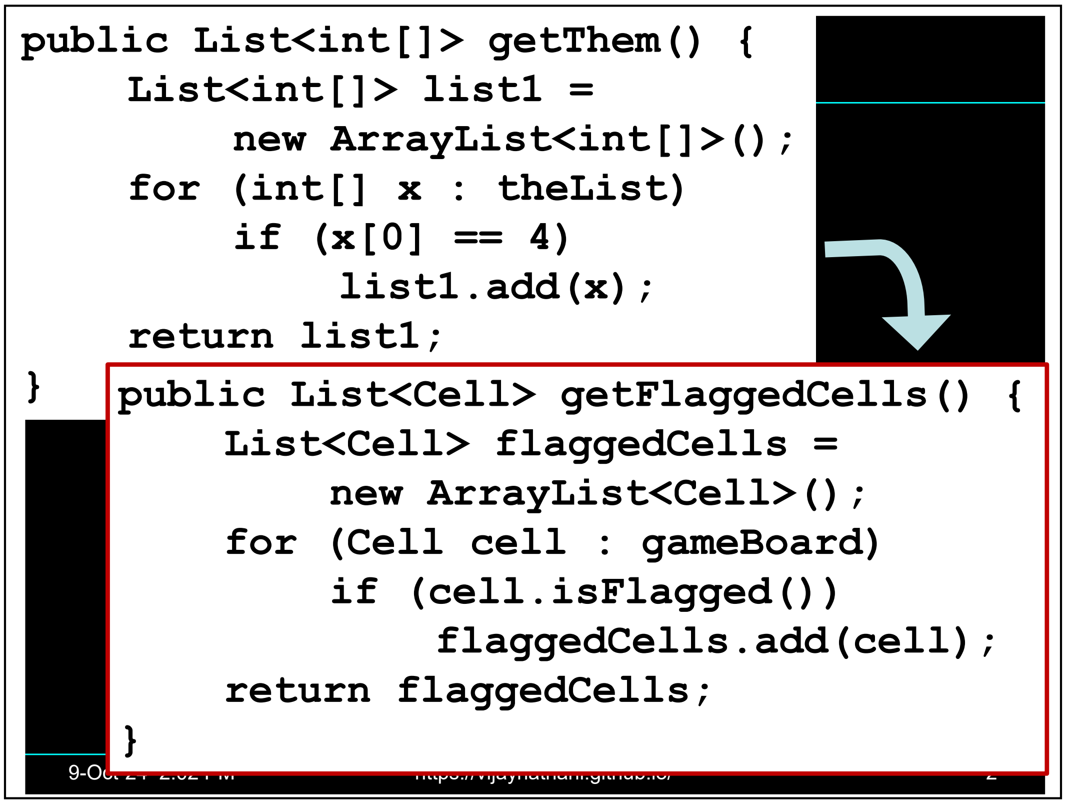


Public API's have to be documented i.e. every class, function, interface, exceptions. Mutable objects that can / cannot be modified.

This is possible by choosing the right variable and function names.

Comments are secondary because they tend to lie

Java: Checkstyle, PMD



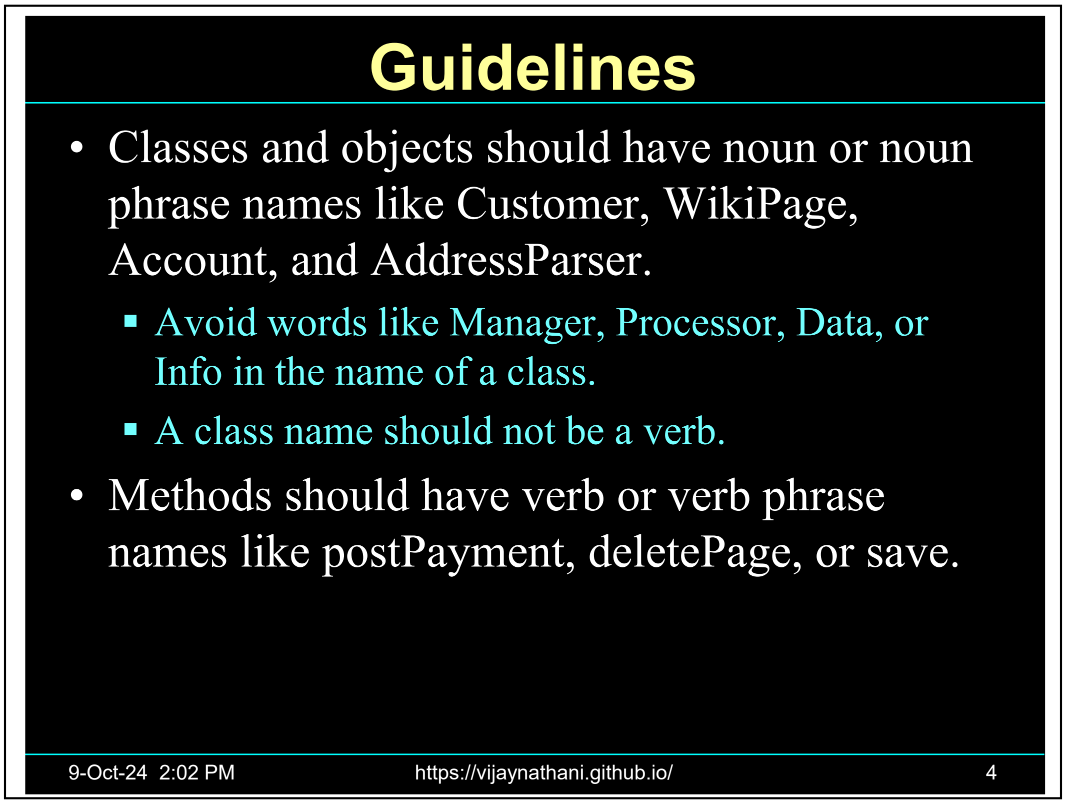
```
public List<int[]> getThem() {
    List<int[]> list1 =
        new ArrayList<int[]>();
    for (int[] x : theList)
        if (x[0] == 4)
            list1.add(x);
    return list1;
}
```

```
public List<Cell> getFlaggedCells() {
    List<Cell> flaggedCells =
        new ArrayList<Cell>();
    for (Cell cell : gameBoard)
        if (cell.isFlagged())
            flaggedCells.add(cell);
    return flaggedCells;
}
```

9-Oct-24 2:02 PM https://vijaynathani.github.io/ 2

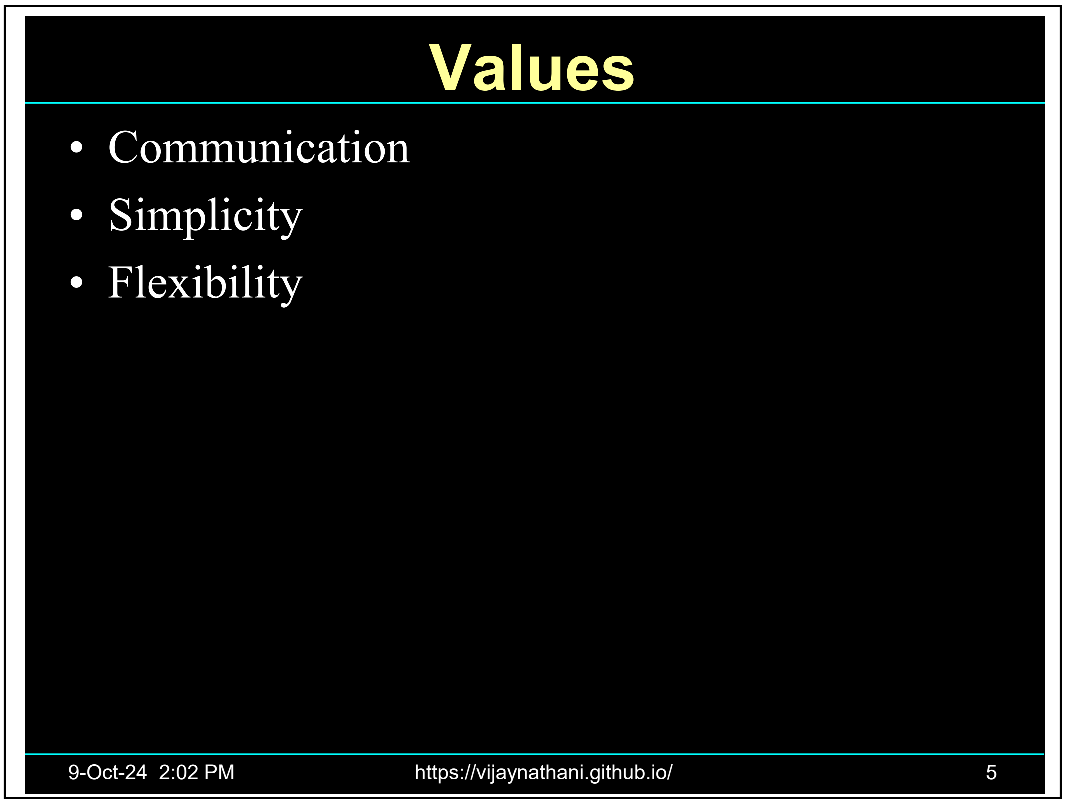
```
class DtaRcrd102 {
    private Date genymdhms;
    private Date modymdhms;
    private final String pszqint = "102";
    /* ... */
};
```

```
class Customer {
    private Date generationTimestamp;
    private Date modificationTimestamp;;
    private static final String
        RECORD_ID = "102";
    /* ... */
};
```



Guidelines

- Classes and objects should have noun or noun phrase names like Customer, WikiPage, Account, and AddressParser.
  - Avoid words like Manager, Processor, Data, or Info in the name of a class.
  - A class name should not be a verb.
- Methods should have verb or verb phrase names like postPayment, deletePage, or save.



Values

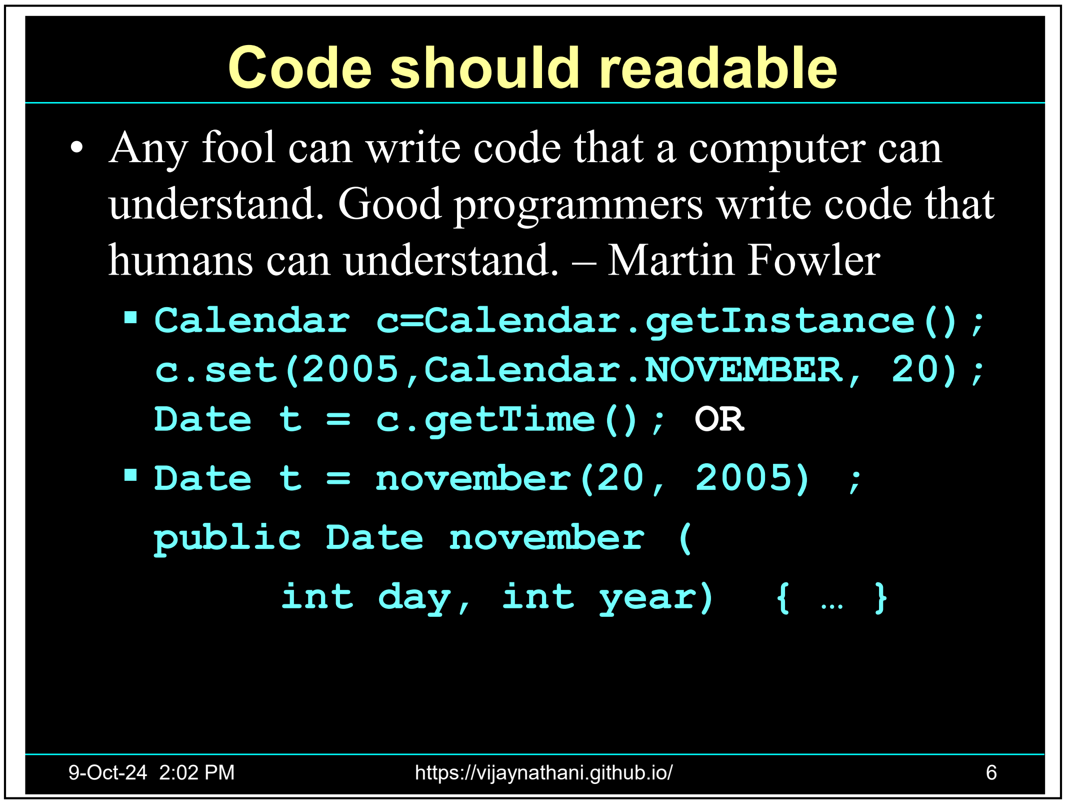
- Communication
- Simplicity
- Flexibility

Mostly they complement each other.

Code communicates well, when a reader can understand it, modify it and use it.

Eliminating excess complexity, makes the program easier to understand, modify and use.

Flexibility means that the program can be changed.

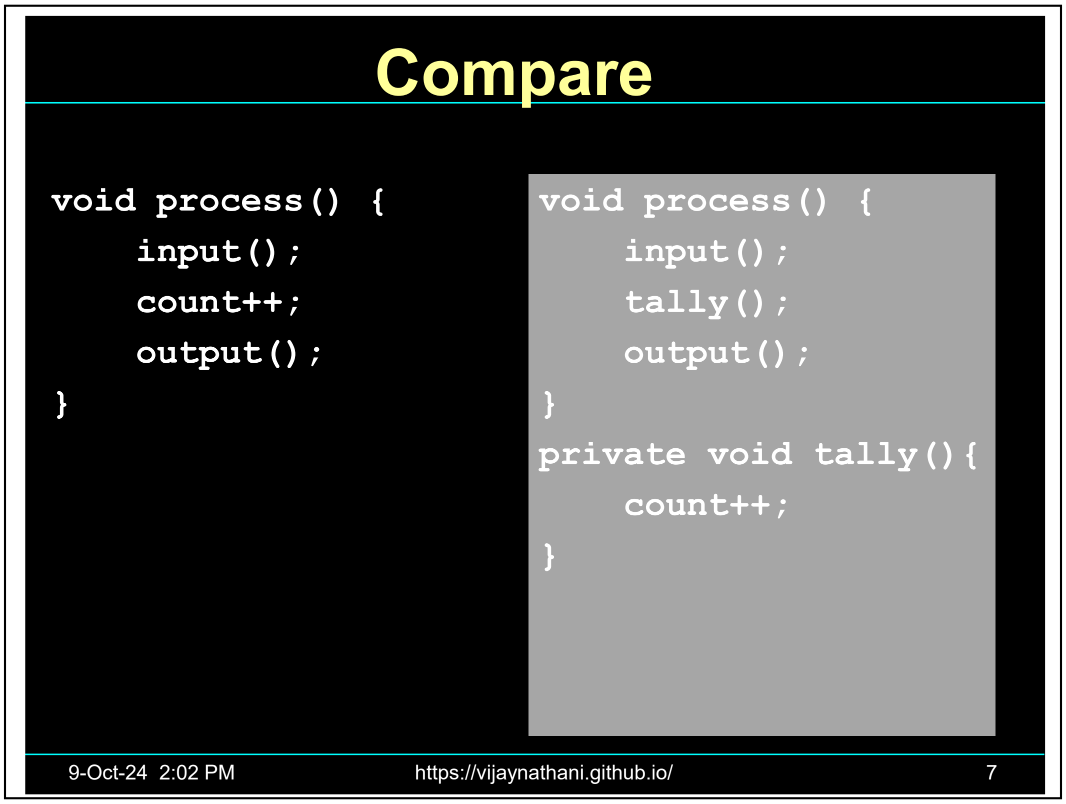


Code should readable

- Any fool can write code that a computer can understand. Good programmers write code that humans can understand. – Martin Fowler

  -
    ```
    Calendar c=Calendar.getInstance();
    c.set(2005,Calendar.NOVEMBER, 20);
    Date t = c.getTime(); OR
    ```

  -
    ```
    Date t = november(20, 2005) ;
    public Date november (
          int day, int year)  { … }
    ```

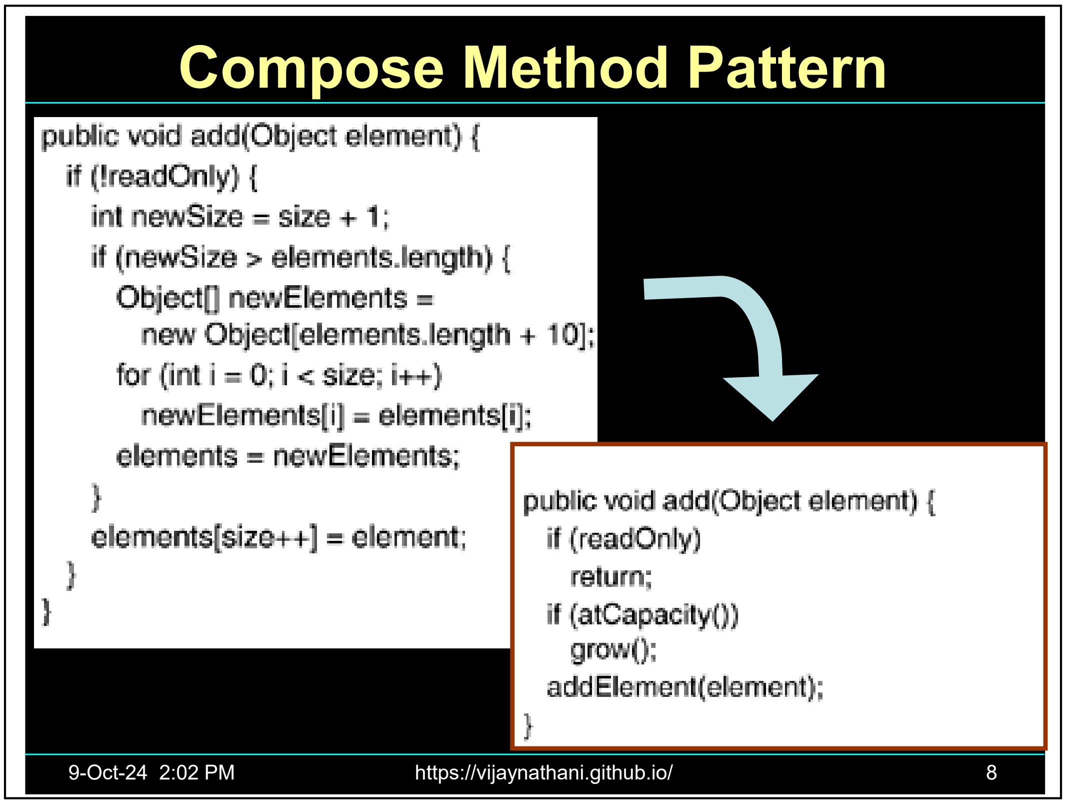


Compare

```
void process() {
    input();
    count++;
    output();
}
```

```
void process() {
    input();
    tally();
    output();
}
private void tally(){
    count++;
}
```

9-Oct-24 2:02 PM https://vijaynathani.github.io/ 7



Compose Method Pattern

```
public void add(Object element) {
  if (!readOnly) {
    int newSize = size + 1;
    if (newSize > elements.length) {
      Object[] newElements =
        new Object[elements.length + 10];
      for (int i = 0; i < size; i++)
        newElements[i] = elements[i];
      elements = newElements;
    }
    elements[size++] = element;
  }
}
```

```
public void add(Object element) {
  if (readOnly)
    return;
  if (atCapacity())
    grow();
  addElement(element);
}
```

9-Oct-24 2:02 PM https://vijaynathani.github.io/ 8

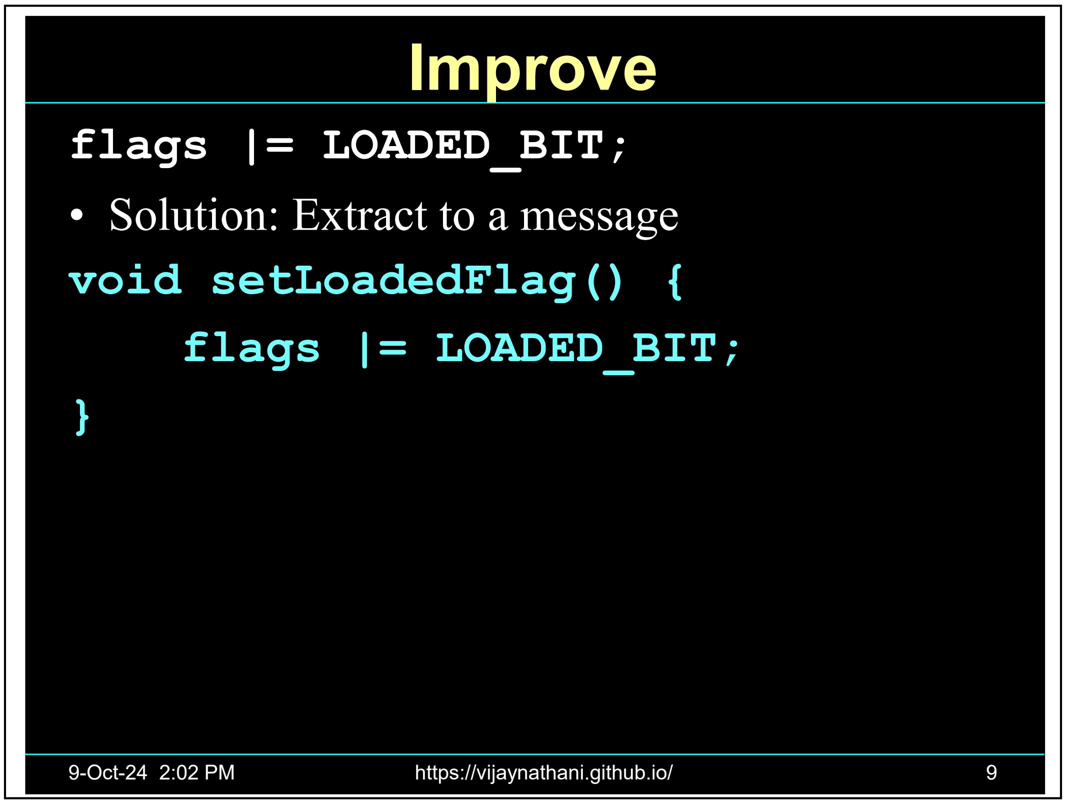
Benefits and Liabilities

+Efficiently communicates what a method does and how it does what it does.

+Simplifies a method by breaking it up into well-named chunks of behavior at the same level of detail.

–Can lead to an overabundance of small methods.

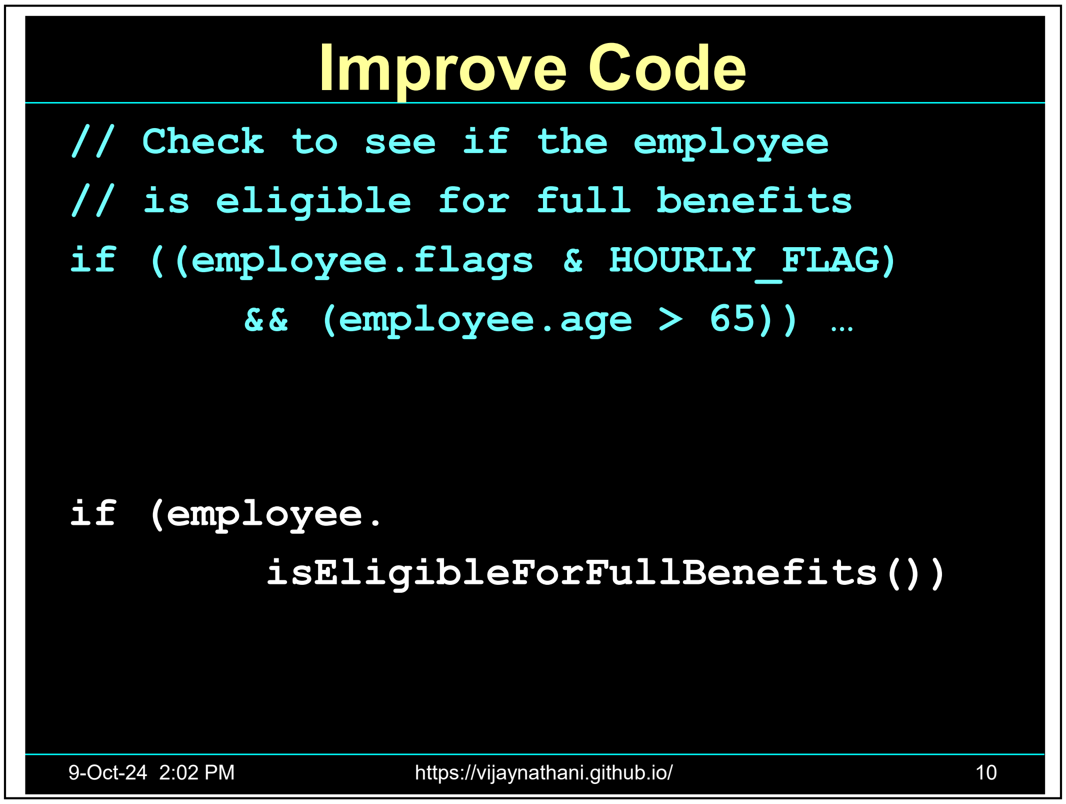
–Can make debugging difficult because logic is spread out across many small methods.


Improve

```
flags |= LOADED_BIT;
```

- Solution: Extract to a message

```
void setLoadedFlag() {
    flags |= LOADED_BIT;
}
```



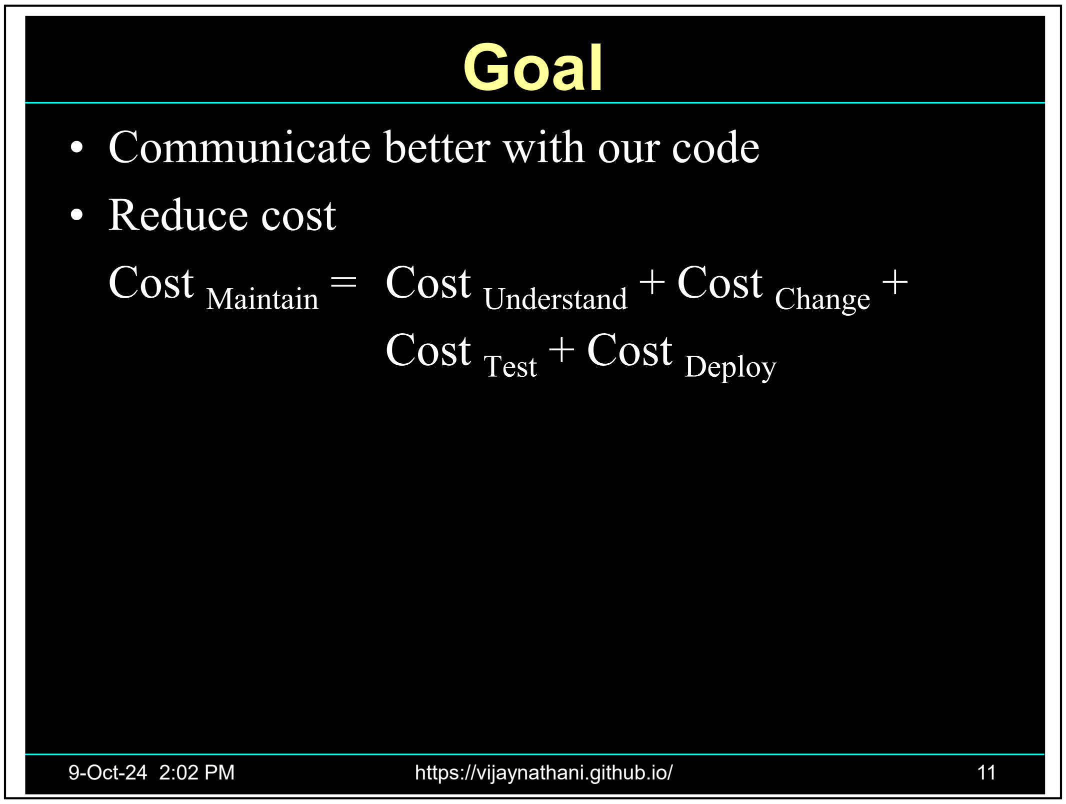
Improve Code

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



if (employee.
        isEligibleForFullBenefits())
```

9-Oct-24 2:02 PM https://vijaynathani.github.io/ 10



Goal

- Communicate better with our code
- Reduce cost

$$\text{Cost}_{\text{Maintain}} = \text{Cost}_{\text{Understand}} + \text{Cost}_{\text{Change}} + \text{Cost}_{\text{Test}} + \text{Cost}_{\text{Deploy}}$$

9-Oct-24 2:02 PM https://vijaynathani.github.io/ 11

Falls between design patterns and Java language manual

Isolate the concurrent portions of the code.

Most programs follow a small set of laws:

•Programs are read more often than they are written

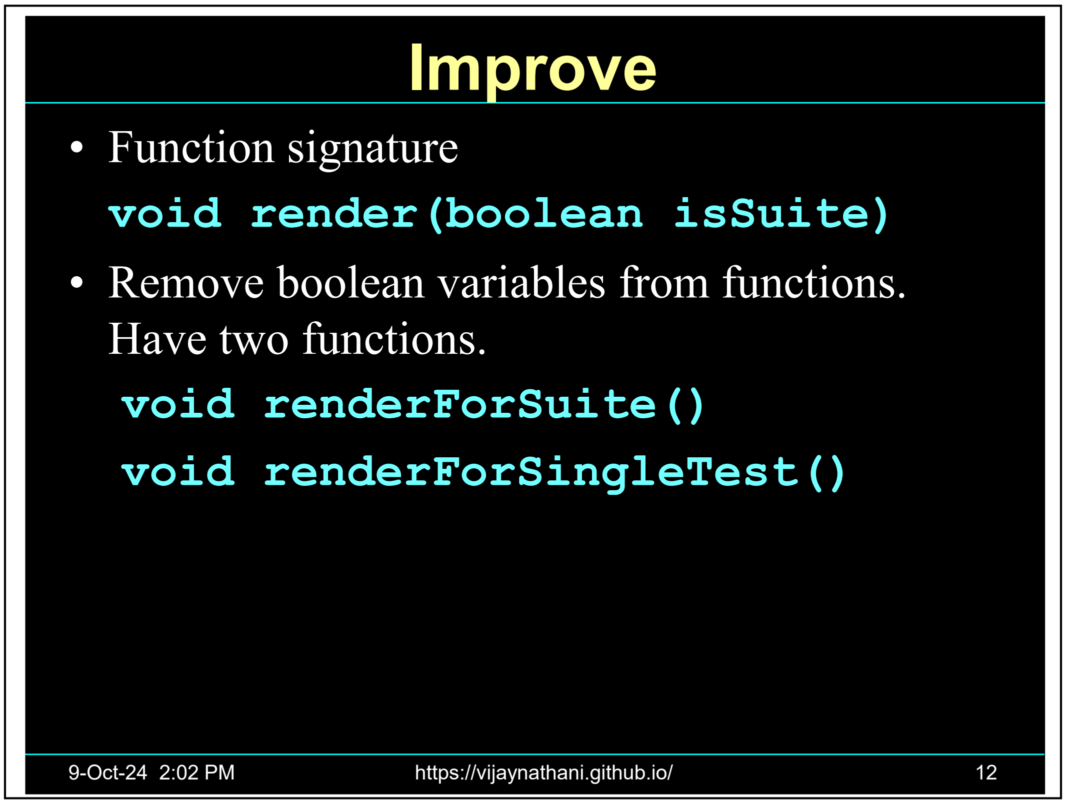
•There is no such thing as “done”. Much more investment will be spent in modifying programs than in developing them initially.

•They are structured using a basic set of state and control flow concepts

•Readers need to understand programs in detail and in concept

Cost to understand code is high. So maintenance is costly. Code will need to change in unanticipated ways.

When code is clear we have fewer defects and smoother development also.

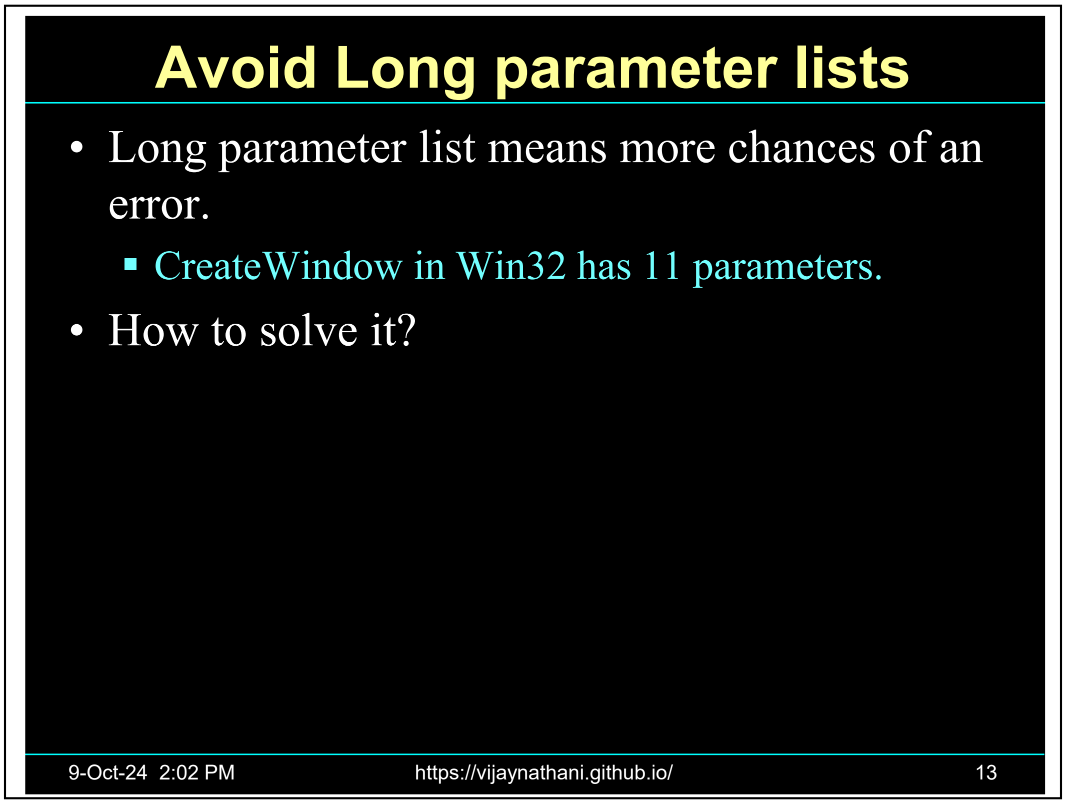

Improve

- Function signature

```
void render(boolean isSuite)
```

- Remove boolean variables from functions. Have two functions.

```
void renderForSuite()
void renderForSingleTest()
```



Avoid Long parameter lists

- Long parameter list means more chances of an error.
  - CreateWindow in Win32 has 11 parameters.
- How to solve it?

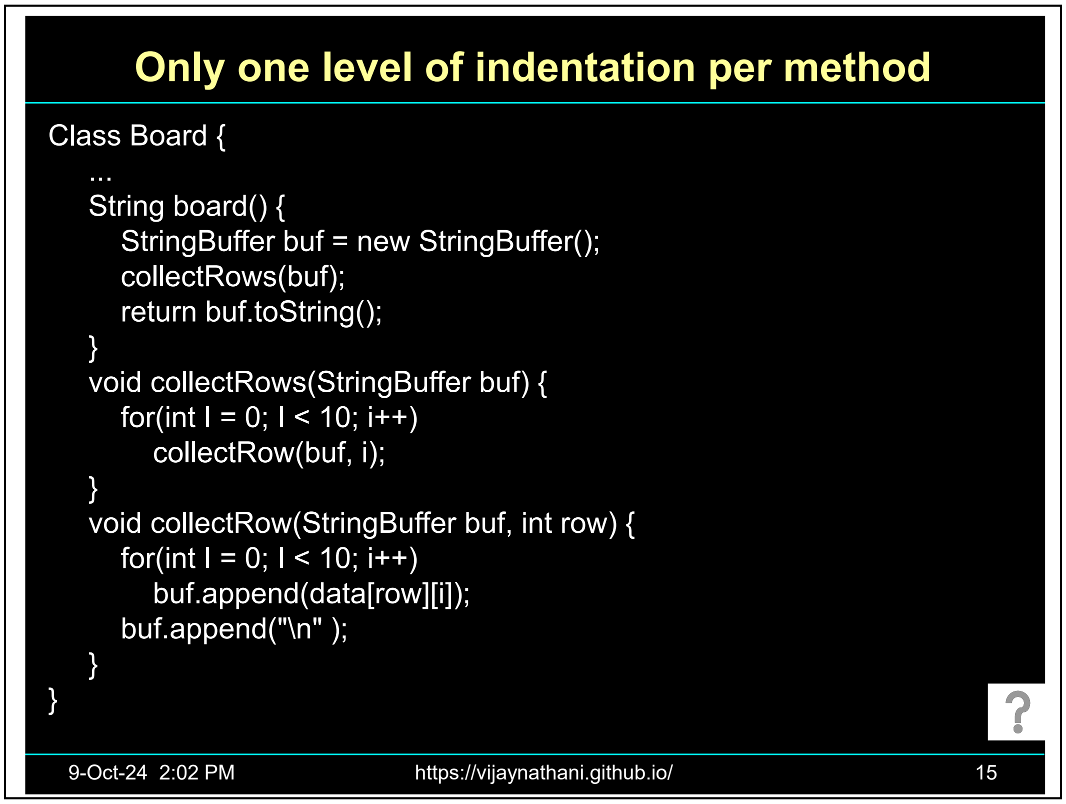
9-Oct-24 2:02 PM https://vijaynathani.github.io/ 13

Break the method or

Create Helper class to hold parameters

Improve

```
class Board {
    ...
    String board() {
        StringBuffer buf = new StringBuffer();
        for(int i = 0; i < 10; i++) {
            for(int j = 0; j < 10; j++)
                buf.append(data[i][j]);
            buf.append("\n" );
        }
        return buf.toString();
    }
}
```



Only one level of indentation per method

```
Class Board {
    ...
    String board() {
        StringBuffer buf = new StringBuffer();
        collectRows(buf);
        return buf.toString();
    }
    void collectRows(StringBuffer buf) {
        for(int I = 0; I < 10; i++)
            collectRow(buf, i);
    }
    void collectRow(StringBuffer buf, int row) {
        for(int I = 0; I < 10; i++)
            buf.append(data[row][i]);
        buf.append("\n" );
    }
}
```

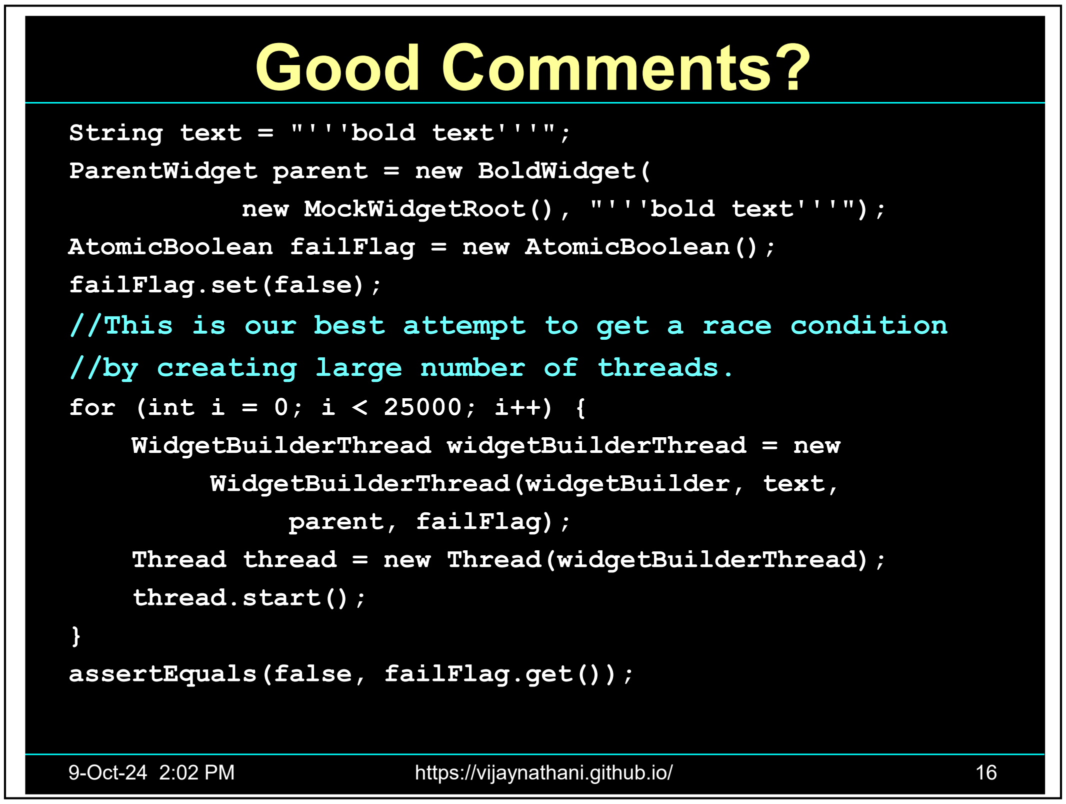
?

9-Oct-24 2:02 PM https://vijaynathani.github.io/ 15

Self documenting code

Q20 – inch; Q06 – NO_GROUPING; Q07 – addHoliday; Q21 – full name in English;

Q22 – complexPassword; Q23 – TokenStream; Q25 - orderItems

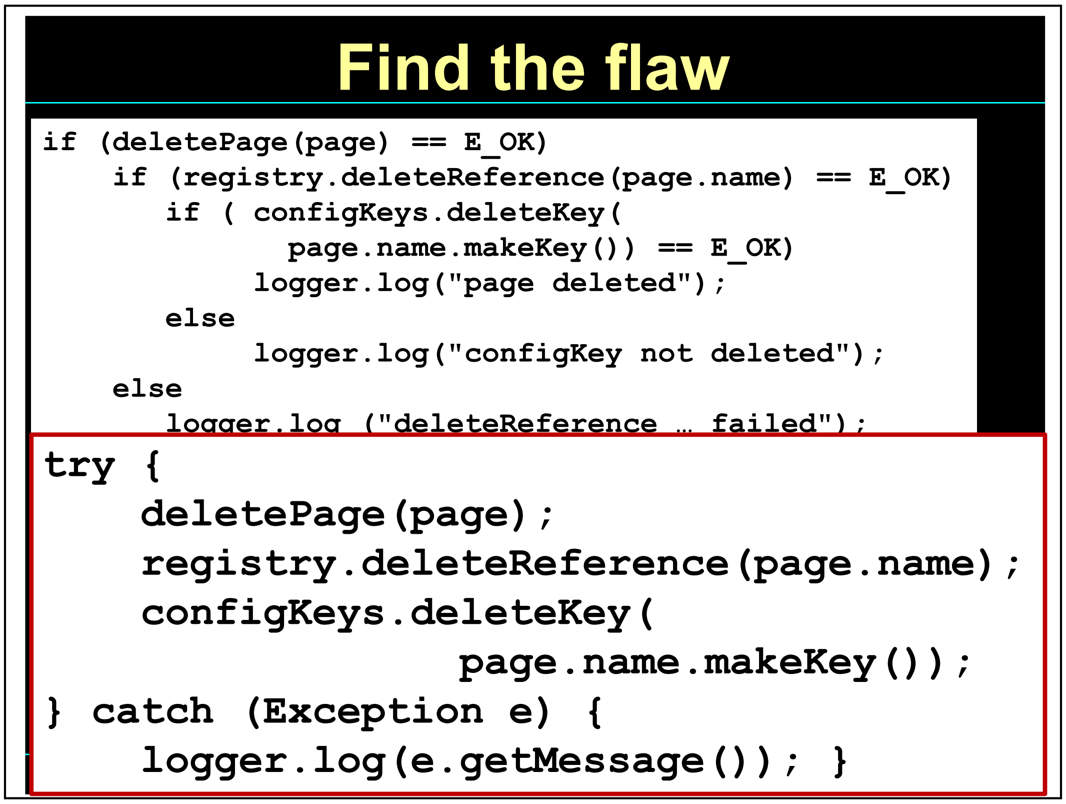


Good Comments?

```
String text = "'''bold text'''";
ParentWidget parent = new BoldWidget(
           new MockWidgetRoot(), "'''bold text'''");
AtomicBoolean failFlag = new AtomicBoolean();
failFlag.set(false);
//This is our best attempt to get a race condition
//by creating large number of threads.
for (int i = 0; i < 25000; i++) {
    WidgetBuilderThread widgetBuilderThread = new
         WidgetBuilderThread(widgetBuilder, text,
              parent, failFlag);
    Thread thread = new Thread(widgetBuilderThread);
    thread.start();
}
assertEquals(false, failFlag.get());
```

9-Oct-24 2:02 PM https://vijaynathani.github.io/ 16

Comment to
• explain WHY we are doing something?
•Also for external documentation. Javadocs public API
•To give warnings: e.g. Don’t run unless you want to kill this program.
•Todo comments



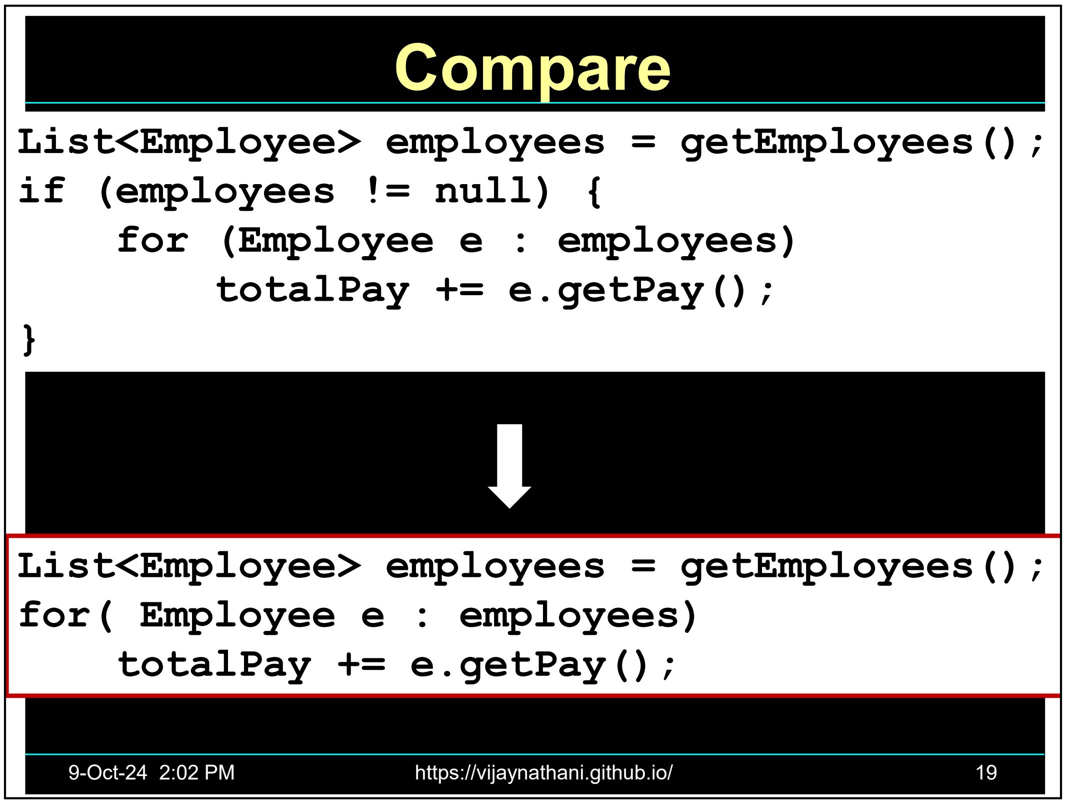
Find the flaw

```
if (deletePage(page) == E_OK)
    if (registry.deleteReference(page.name) == E_OK)
       if ( configKeys.deleteKey(
                page.name.makeKey()) == E_OK)
            logger.log("page deleted");
       else
            logger.log("configKey not deleted");
    else
       logger.log ("deleteReference … failed");
```

```
try {
    deletePage(page);
    registry.deleteReference(page.name);
    configKeys.deleteKey(
                 page.name.makeKey());
} catch (Exception e) {
    logger.log(e.getMessage()); }
```

Samurai Principle

- Throw exception if any error occurs.



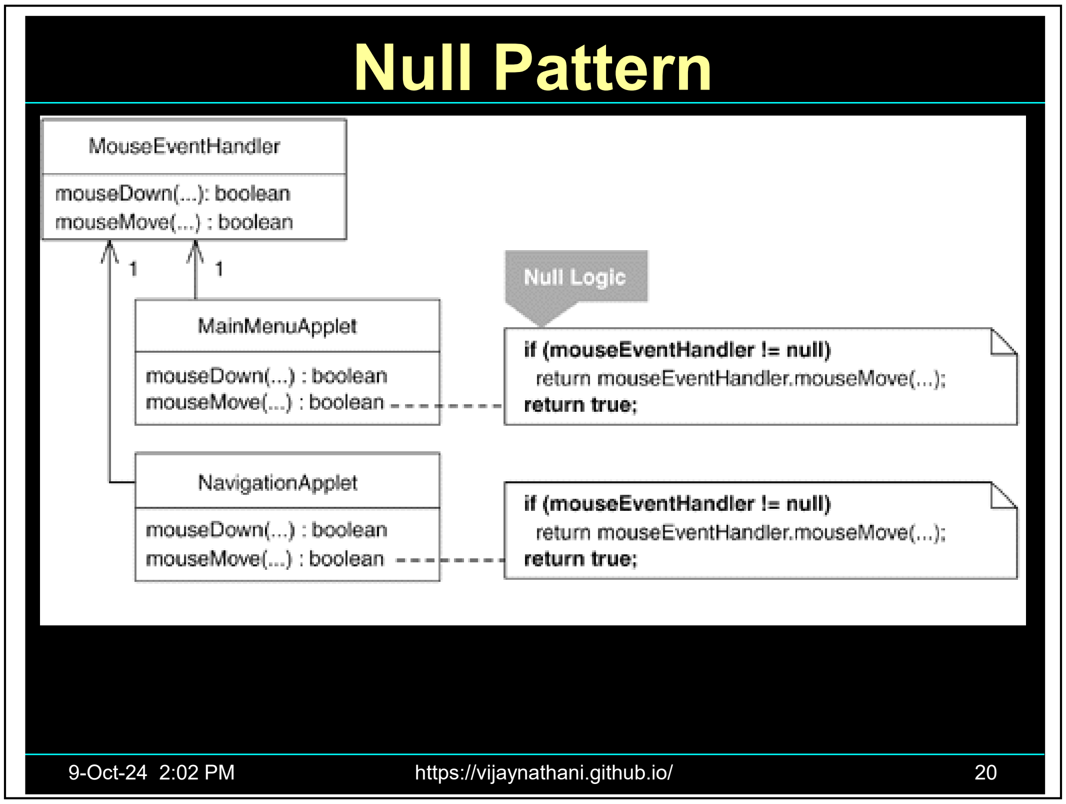
Compare

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

⬇

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

Java has Collections.emptyList() for this. It is immutable.



Null Pattern

```
if (mouseEventHandler != null)
  return mouseEventHandler.mouseMove(...);
return true;
```

```
if (mouseEventHandler != null)
  return mouseEventHandler.mouseMove(...);
return true;
```

9-Oct-24 2:02 PM https://vijaynathani.github.io/ 20

Null pattern

Avoid NullPointerException in code

Return “” instead of null for String class

Return an array with zero elements instead of null.

=======================

Benefits and Liabilities

+ Prevents null errors without duplicating null logic.

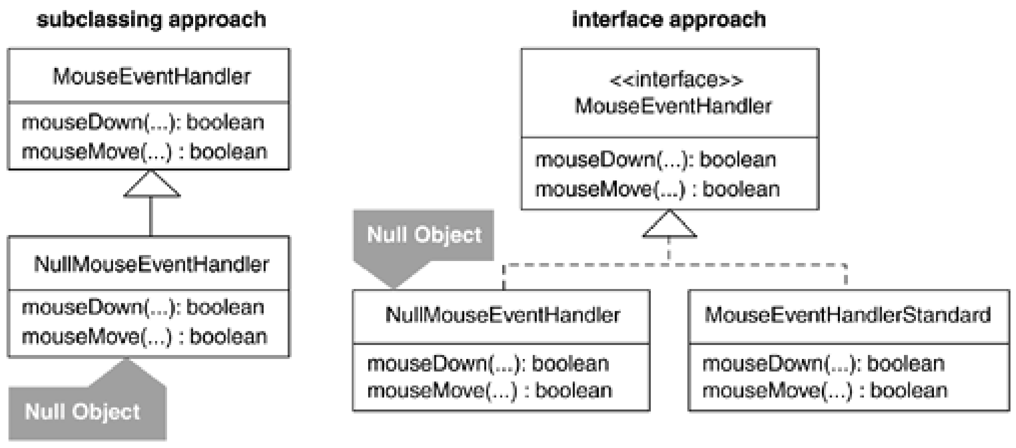
+ Simplifies code by minimizing null tests.

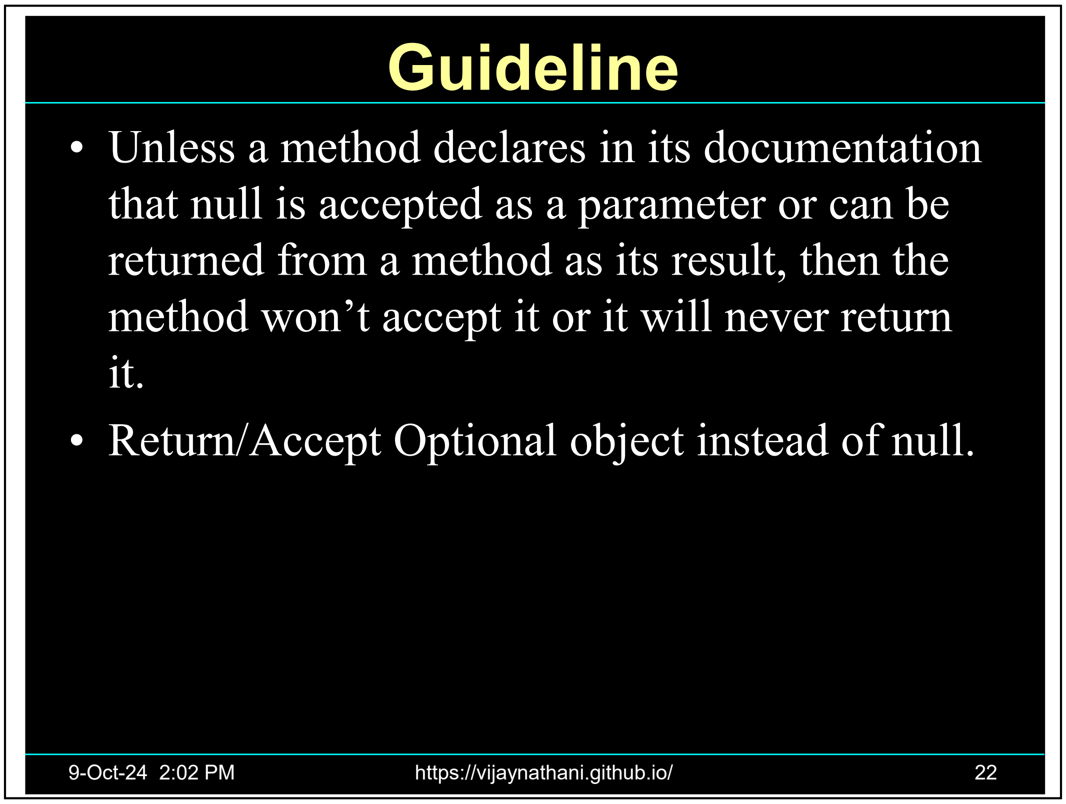
– Complicates a design when a system needs few null tests.

– Can yield redundant null tests if programmers are unaware of a Null Object implementation.

– Complicates maintenance. Null Objects that have a superclass must override all newly inherited public methods.

Null Object by Interface



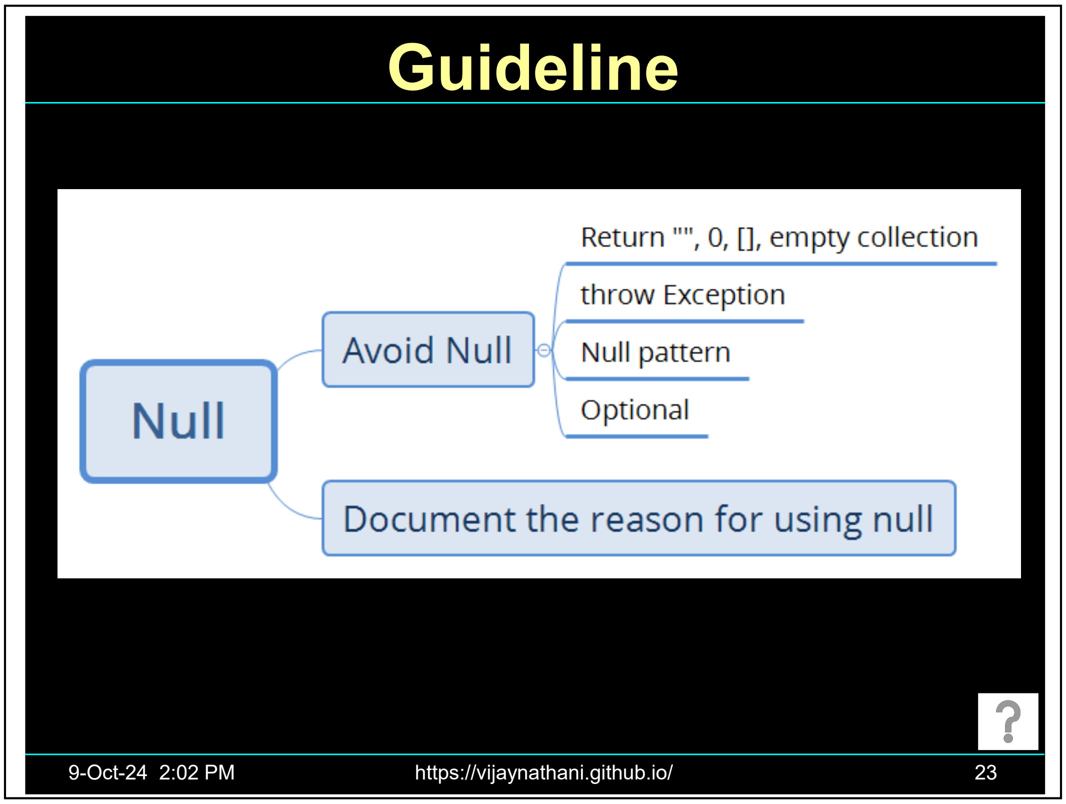


Guideline

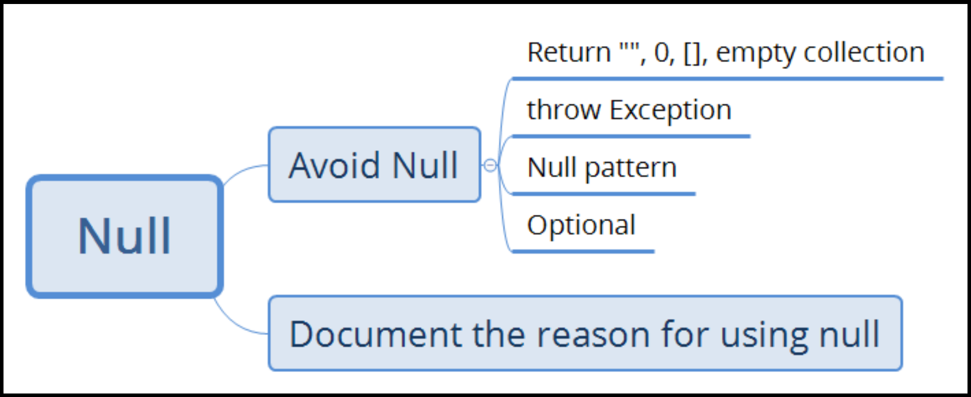
- Unless a method declares in its documentation that null is accepted as a parameter or can be returned from a method as its result, then the method won't accept it or it will never return it.
- Return/Accept Optional object instead of null.

9-Oct-24 2:02 PM https://vijaynathani.github.io/ 22

Optional class is present in Java 8. If you are using older versions of Java, then it is also present in Guava library.

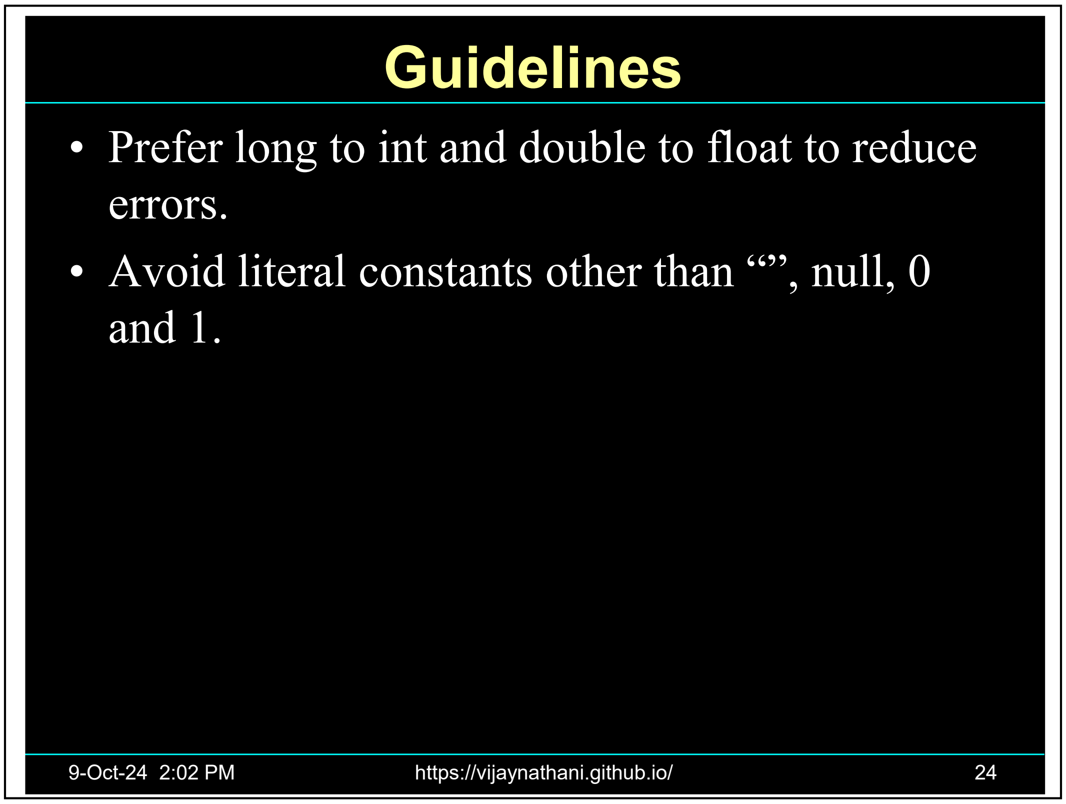


Guideline


Null
Avoid Null
Return "", 0, [], empty collection
throw Exception
Null pattern
Optional
Document the reason for using null

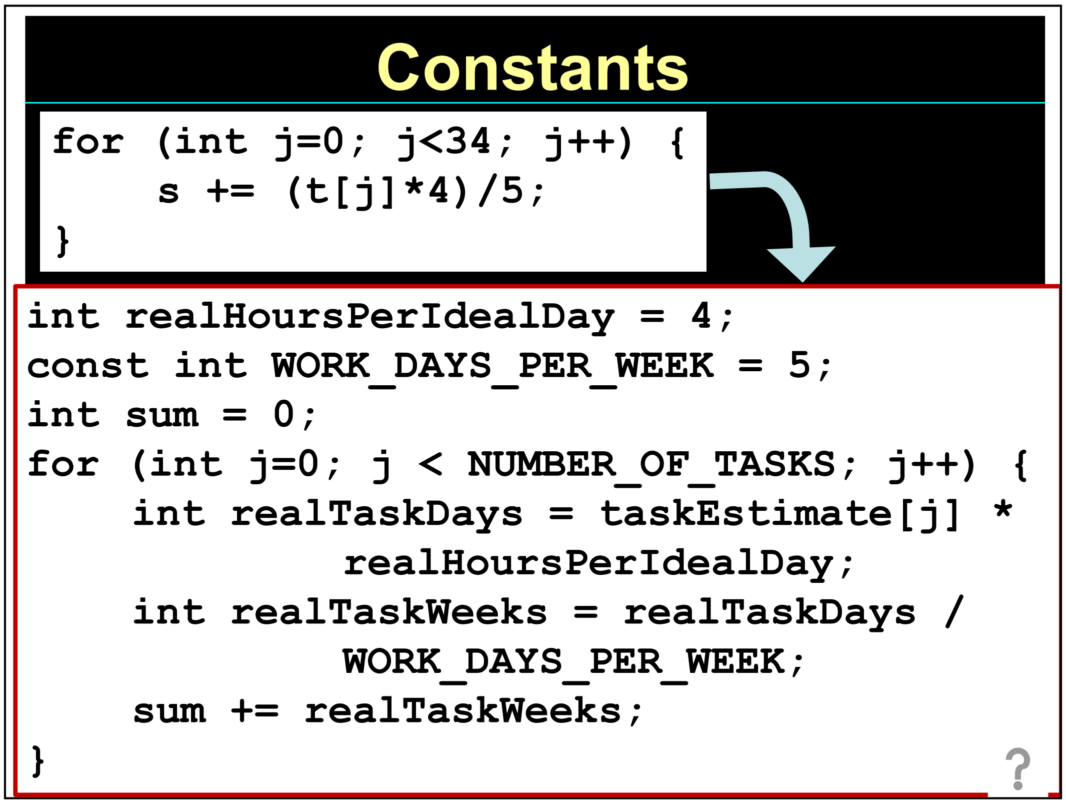
9-Oct-24 2:02 PM https://vijaynathani.github.io/ 23

Q28Artists



Guidelines

- Prefer long to int and double to float to reduce errors.
- Avoid literal constants other than “”, null, 0 and 1.



Constants

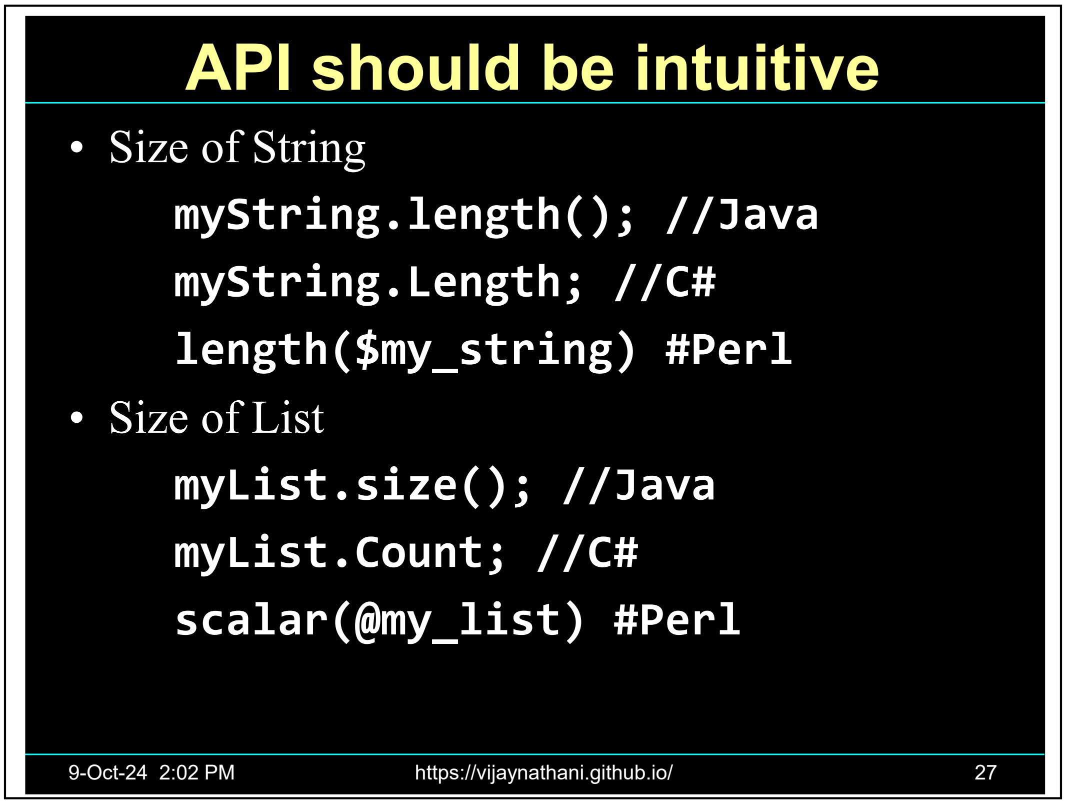
```
for (int j=0; j<34; j++) {
     s += (t[j]*4)/5;
}
```

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

Q32 – FoodSalesReport.

Guidelines

- An interface should be designed so that it is easy to use and difficult to misuse.



API should be intuitive

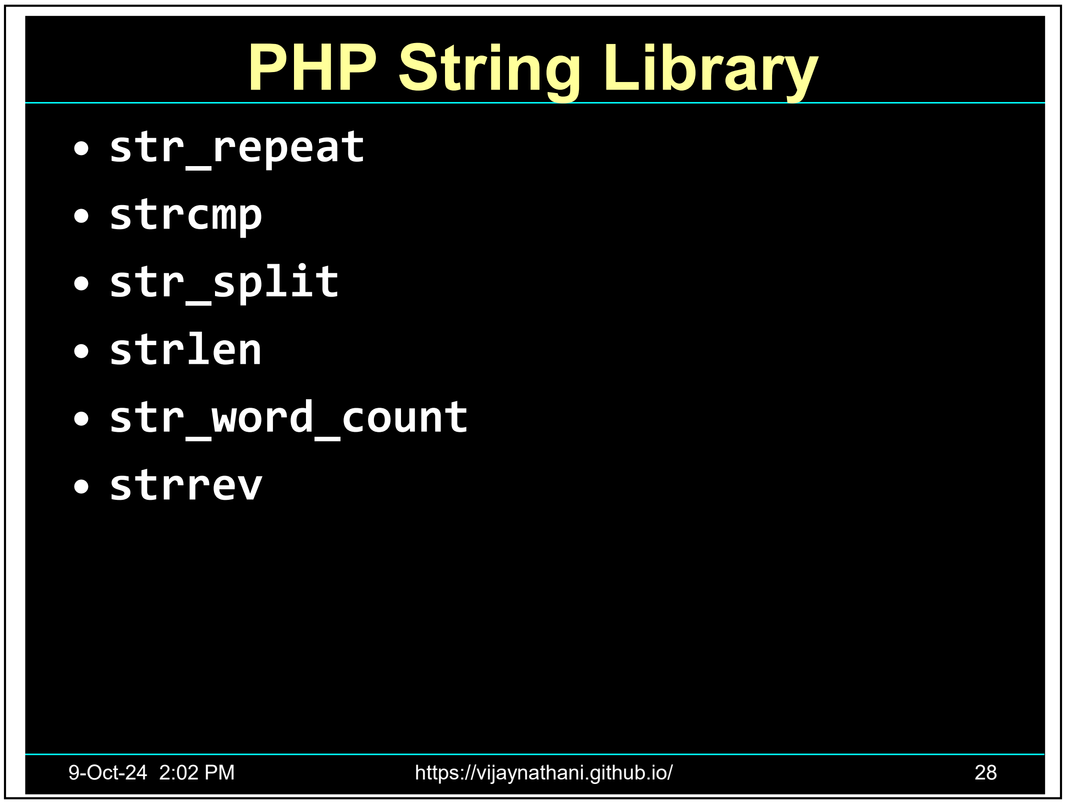
- Size of String

```
myString.length(); //Java
myString.Length; //C#
length($my_string) #Perl
```

- Size of List

```
myList.size(); //Java
myList.Count; //C#
scalar(@my_list) #Perl
```

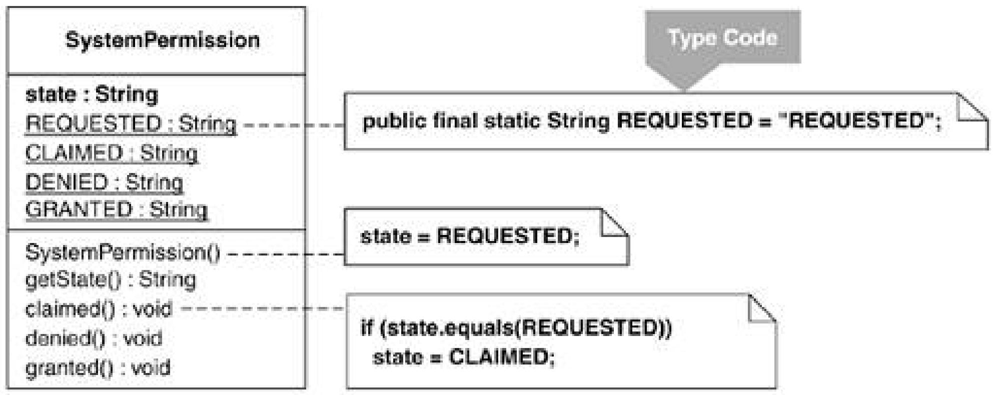
9-Oct-24 2:02 PM https://vijaynathani.github.io/ 27



PHP String Library

- `str_repeat`
- `strcmp`
- `str_split`
- `strlen`
- `str_word_count`
- `strrev`

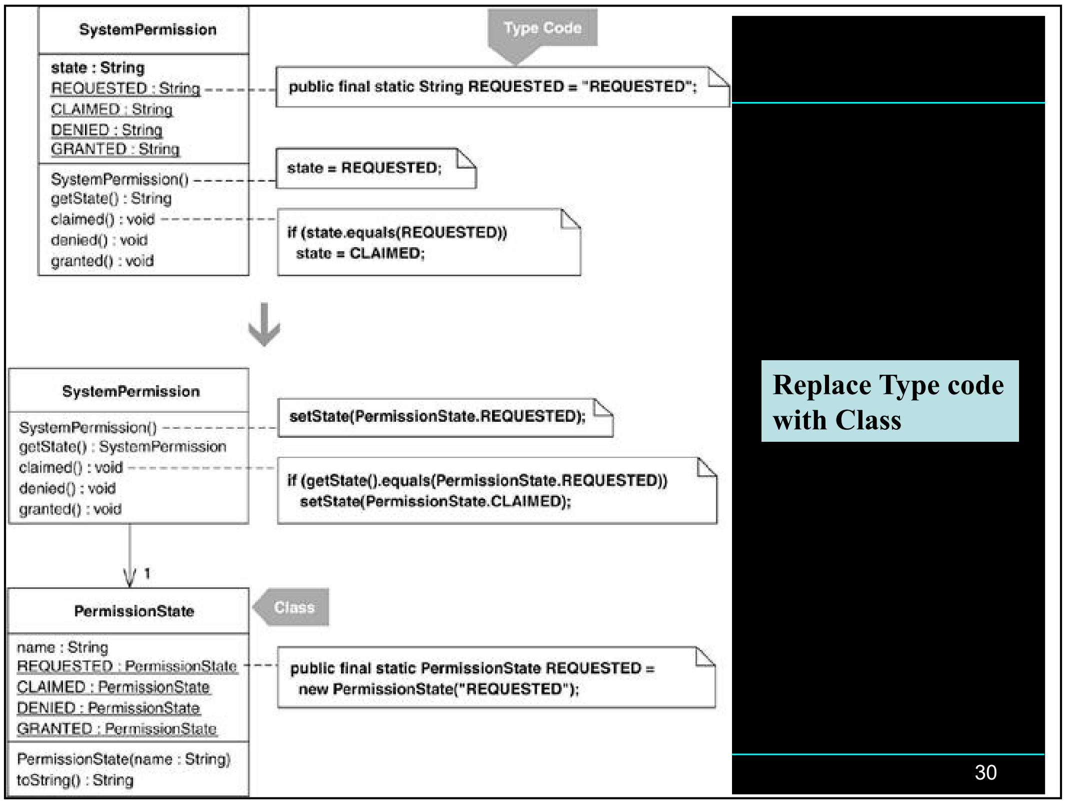
Improve Code



```
public final static String REQUESTED = "REQUESTED";
```

```
state = REQUESTED;
```

```
if (state.equals(REQUESTED))
  state = CLAIMED;
```



```
public final static String REQUESTED = "REQUESTED";
```

```
state = REQUESTED;
```

```
if (state.equals(REQUESTED))
  state = CLAIMED;
```

```
setState(PermissionState.REQUESTED);
```

```
if (getState().equals(PermissionState.REQUESTED))
  setState(PermissionState.CLAIMED);
```

```
public final static PermissionState REQUESTED =
  new PermissionState("REQUESTED");
```

Replace Type code with Class

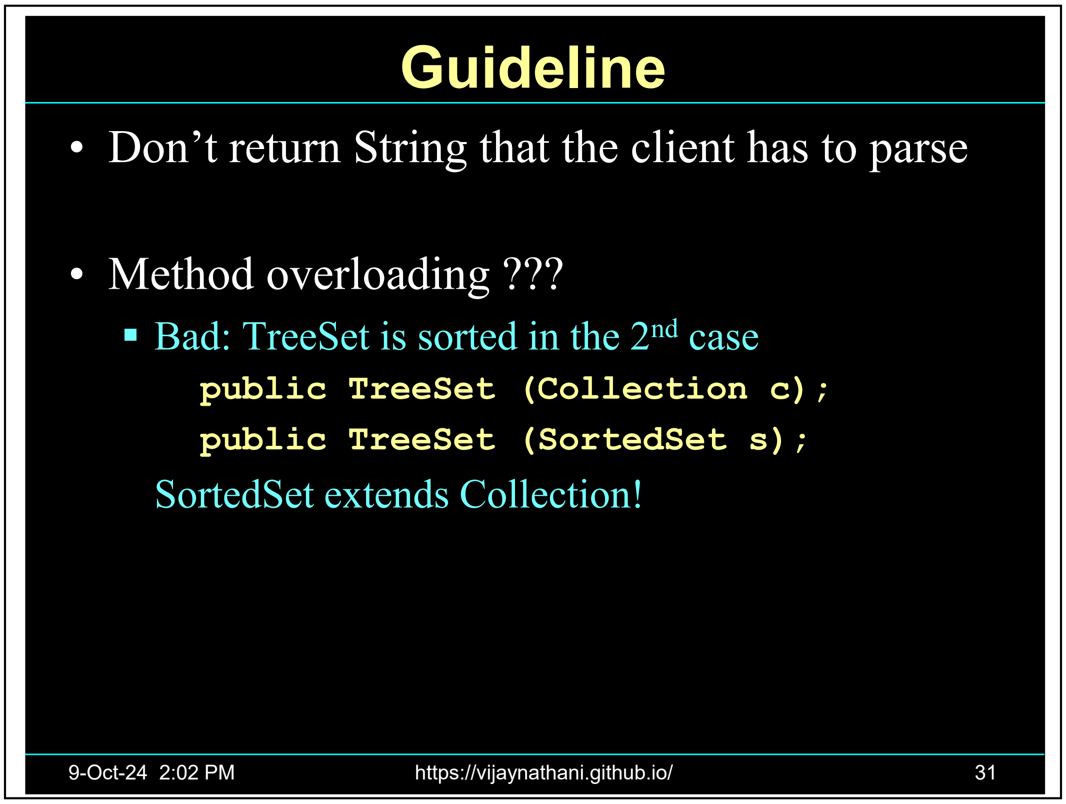
A field's type (e.g., a String or int) fails to protect it from unsafe assignments and invalid equality comparisons.

Constrain the assignments and equality comparisons by making the type of the field a class.

Benefits and Liabilities

+ Provides better protection from invalid assignments and comparisons.

– Requires more code than using unsafe type does.



Guideline

- Don’t return String that the client has to parse
- Method overloading ???
  - Bad: TreeSet is sorted in the 2nd case

```
public TreeSet (Collection c);
public TreeSet (SortedSet s);
```

SortedSet extends Collection!

9-Oct-24 2:02 PM https://vijaynathani.github.io/ 31

Return of sting with multiple values is bad because

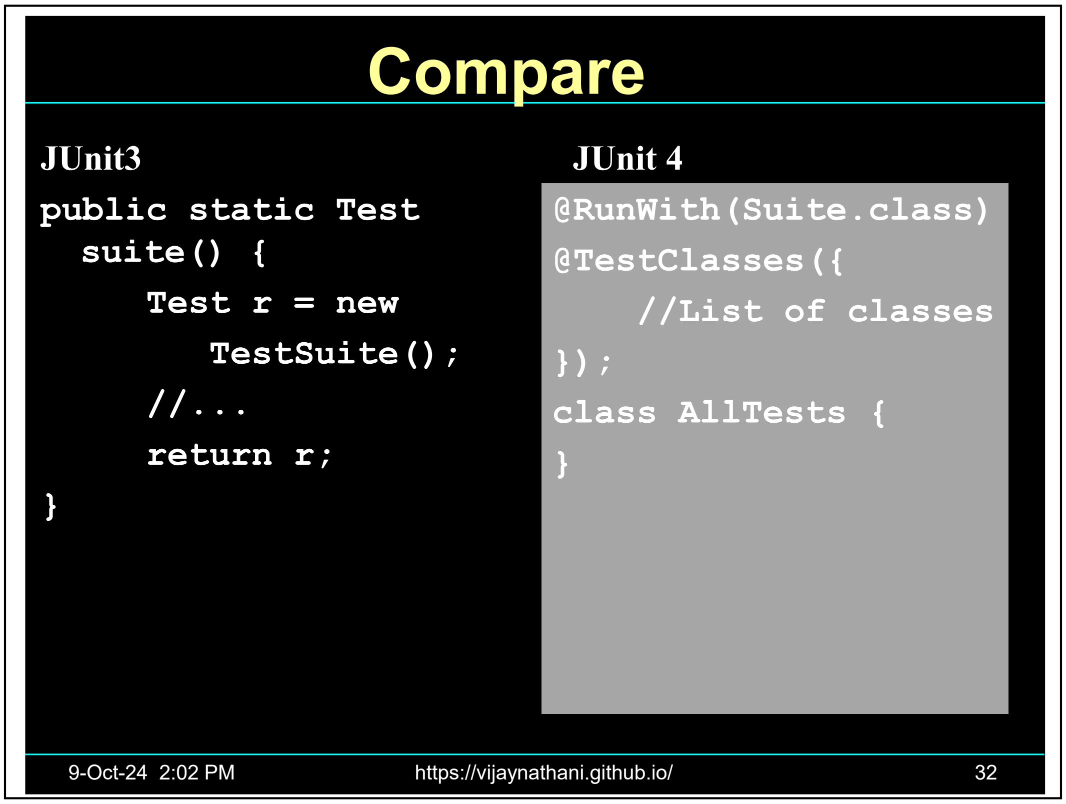
Future modifications will become difficult

Bad: In Java, printStackTrace of Throwable class

==

Overloading - Avoid same name for multiple methods with same number of arguments

In C# and C++, operator overloading should be used only when it is clear. It should not be very frequent.



Compare

JUnit3

```
public static Test
  suite() {
     Test r = new
        TestSuite();
     //...
     return r;
}
```

JUnit 4

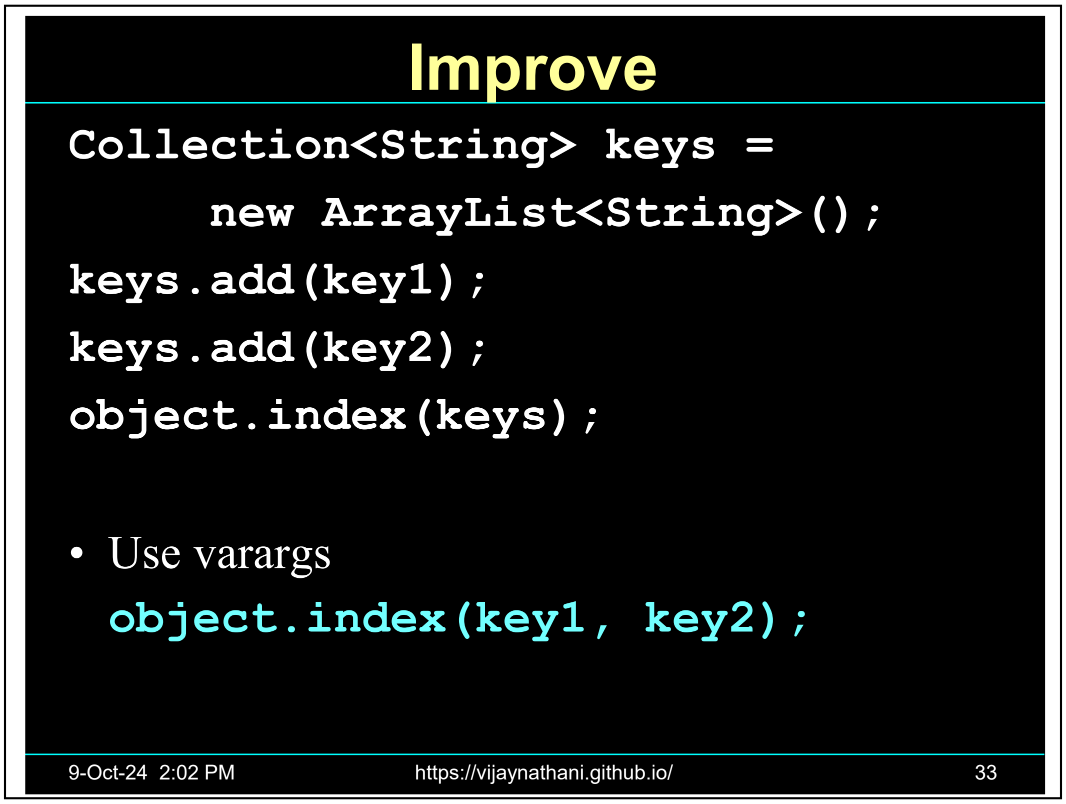
```
@RunWith(Suite.class)
@TestClasses({
    //List of classes
});
class AllTests {
}
```

9-Oct-24 2:02 PM https://vijaynathani.github.io/ 32

The code on the right

•Is flexible because any runner can be used

•Less prone to error, because the caller does not have to worry about exceptions

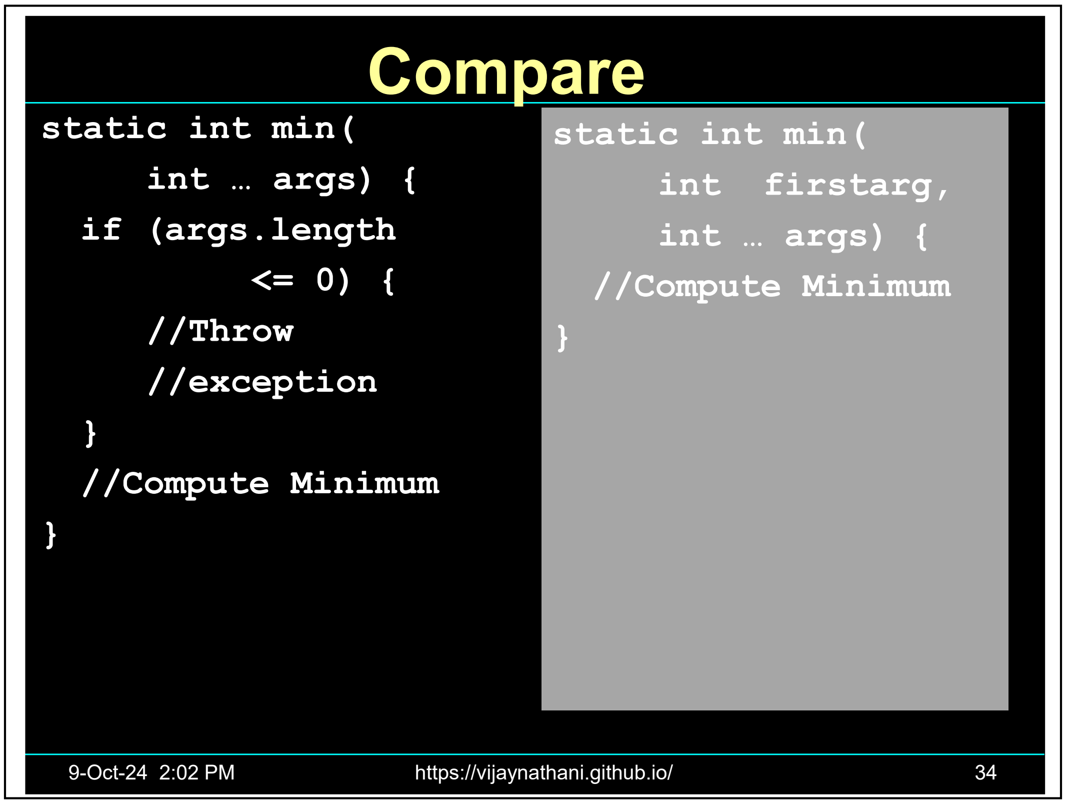


Improve

```
Collection<String> keys =
     new ArrayList<String>();
keys.add(key1);
keys.add(key2);
object.index(keys);
```

- Use varargs

```
object.index(key1, key2);
```



Compare

```
static int min(
     int … args) {
  if (args.length
          <= 0) {
     //Throw
     //exception
  }
  //Compute Minimum
}
```

```
static int min(
     int  firstarg,
     int … args) {
  //Compute Minimum
}
```

9-Oct-24 2:02 PM https://vijaynathani.github.io/ 34

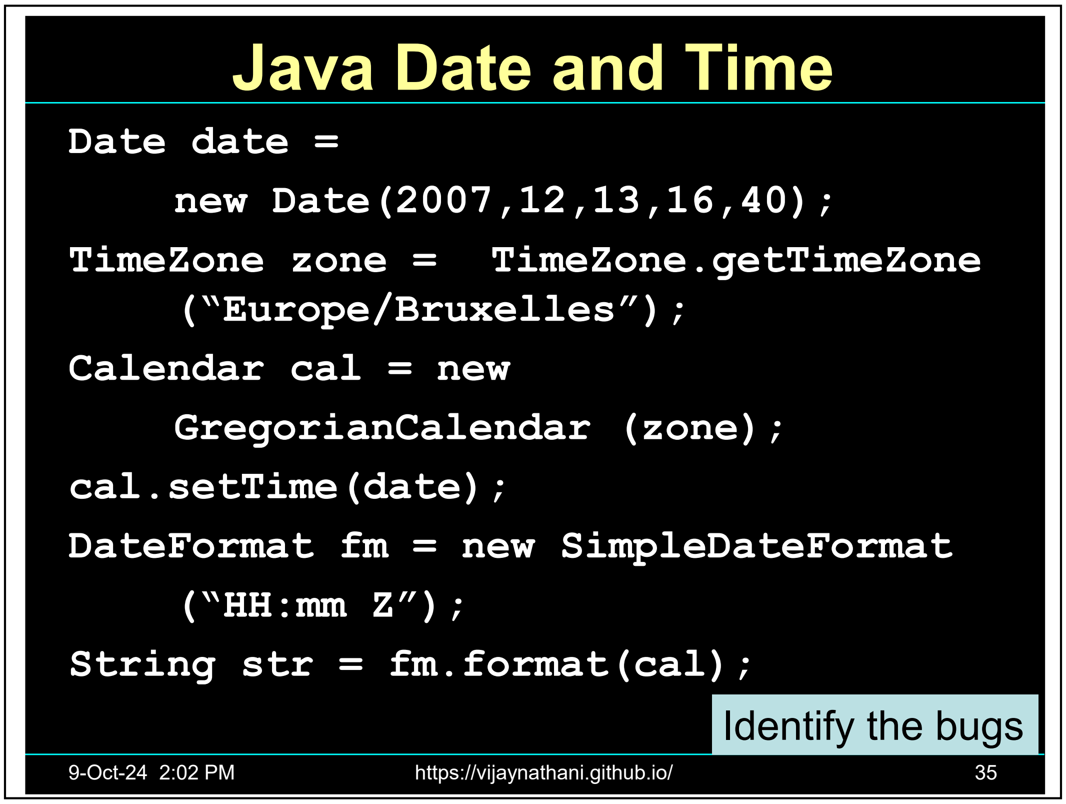
The code on the right

•Is flexible because any runner can be used

•Less prone to error, because the caller does not have to worry about exceptions

In C#, the syntax is

static int min(int firstarg, params int[] args)



Java Date and Time

```
Date date =
    new Date(2007,12,13,16,40);
TimeZone zone =  TimeZone.getTimeZone
     ("Europe/Bruxelles");
Calendar cal = new
    GregorianCalendar (zone);
cal.setTime(date);
DateFormat fm = new SimpleDateFormat
     ("HH:mm Z");
String str = fm.format(cal);
```

Identify the bugs

9-Oct-24 2:02 PM https://vijaynathani.github.io/ 35

Bug 1: Year should be 107, since base is 1900.

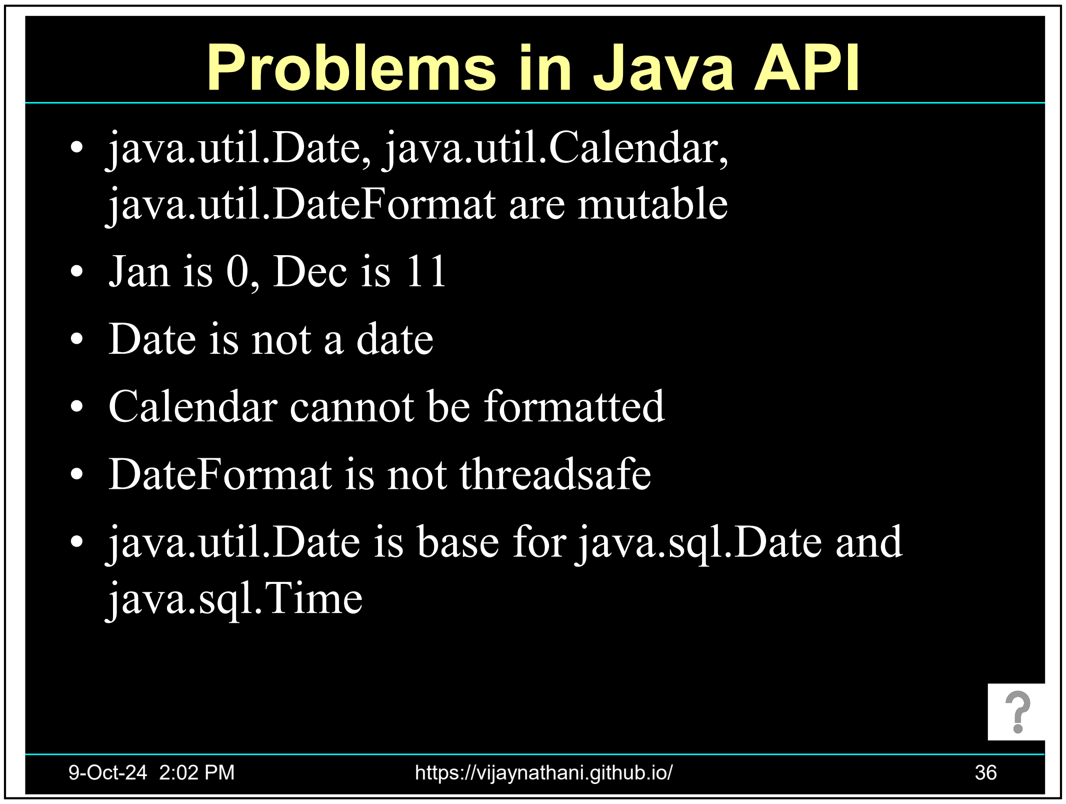
Bug 2: Month should be 11 instead of 12. Since Jan is 0.

Bug 3: “Europe/Bruxelles”. Bruzelles is capital of Brussels. It is capital of Belgium. Different people pronounce it different ways. Java returns GMT.

Bug 4: We are creating cal object with invalid or wrong value of date.

Not sure - Bug 5: fm.format gives runtime exception because it cannot format calendar. It can format only dates. So we need to call get “cal.getTime()” and pass the returned date to “fm.format”

Not sure - Bug 6: We have not set the timezone in DateFormat. It needs to be set before calling format.



Problems in Java API

- java.util.Date, java.util.Calendar, java.util.DateFormat are mutable
- Jan is 0, Dec is 11
- Date is not a date
- Calendar cannot be formatted
- DateFormat is not threadsafe
- java.util.Date is base for java.sql.Date and java.sql.Time

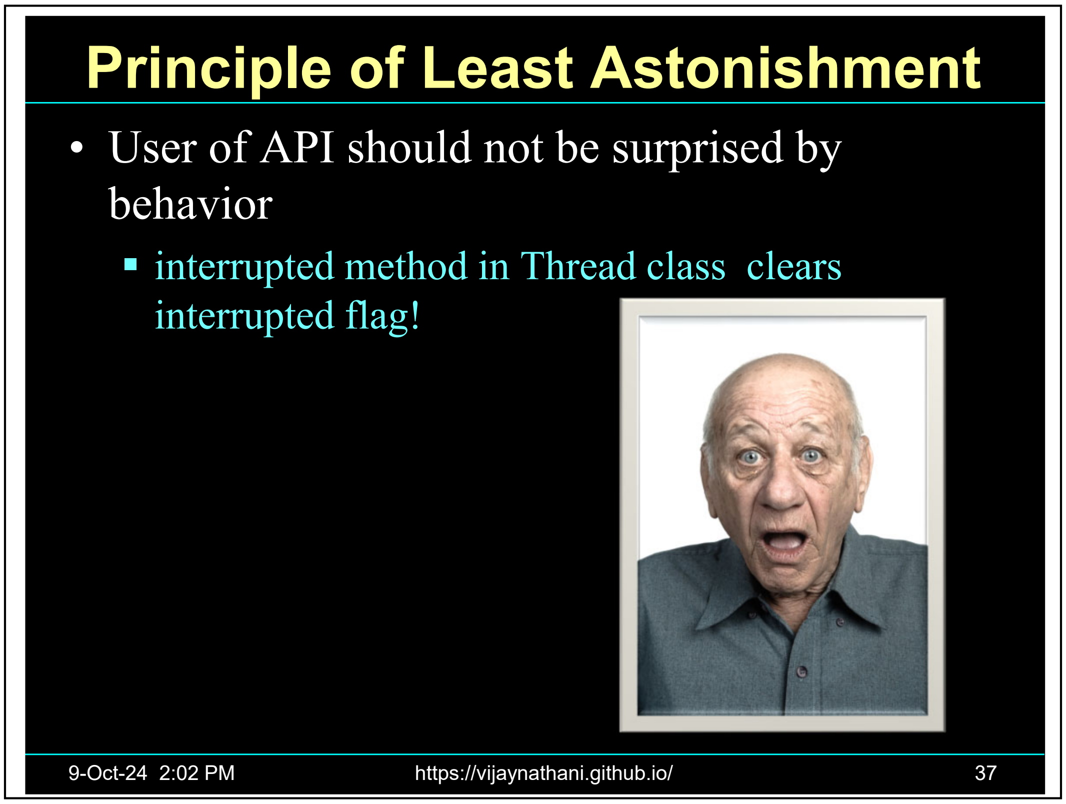
9-Oct-24 2:02 PM https://vijaynathani.github.io/ 36

Q53 – Piano

Q73 – Student, Teacher

Date is not a date because: It has time & It uses from 1900

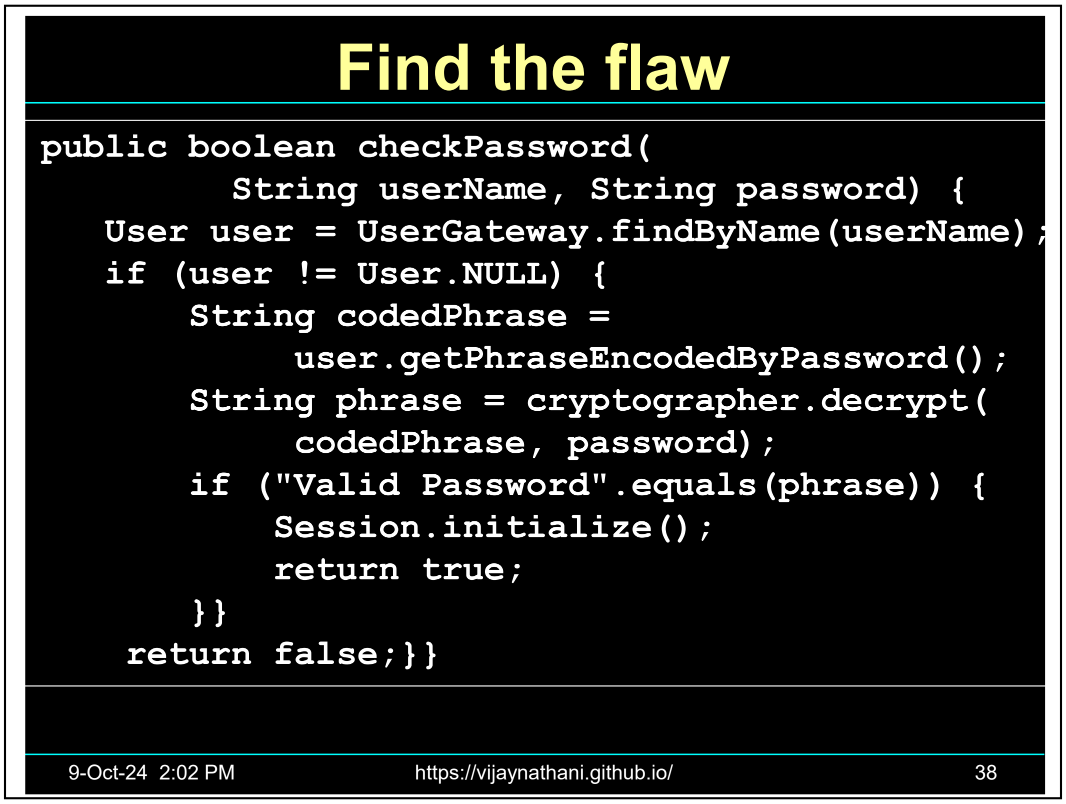
java.util.Date should not base class for java.sql.Date and Time because getYear on java.sql.Time throws an illegal argument exception.



Principle of Least Astonishment

- User of API should not be surprised by behavior
  - interrupted method in Thread class clears interrupted flag!

It should have been called clearInterruptedFlag



Find the flaw

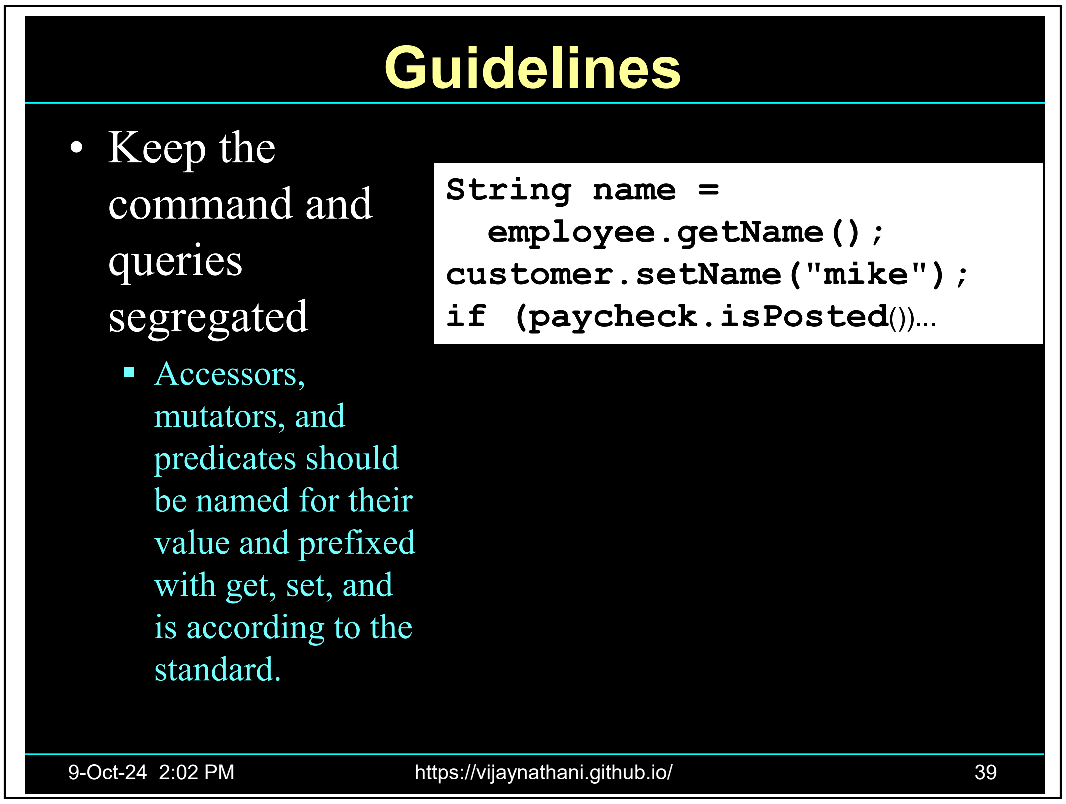
```
public boolean checkPassword(
         String userName, String password) {
   User user = UserGateway.findByName(userName);
   if (user != User.NULL) {
       String codedPhrase =
            user.getPhraseEncodedByPassword();
       String phrase = cryptographer.decrypt(
            codedPhrase,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

9-Oct-24 2:02 PM https://vijaynathani.github.io/ 38

This function uses a standard algorithm to match a userName to a password. It returns true if they match and false if anything goes wrong. But it also has a side effect. Can you spot it?

The side effect is the call to Session.initialize(), of course. The checkPassword function, by its name, says that it checks the password. The name does not imply that it initializes the session. So a caller who believes what the name of the function says runs the risk of erasing the existing session data when he or she decides to check the validity of the user.

This side effect creates a temporal coupling. That is, checkPassword can only be called at certain times (in other words, when it is safe to initialize the session). If it is called out of order, session data may be inadvertently lost. Temporal couplings are confusing, especially when hidden as a side effect. If you must have a temporal coupling, you should make it clear in the name of the function. In this case we might rename the function checkPasswordAndInitializeSession, though that certainly violates "Do one thing."



Guidelines

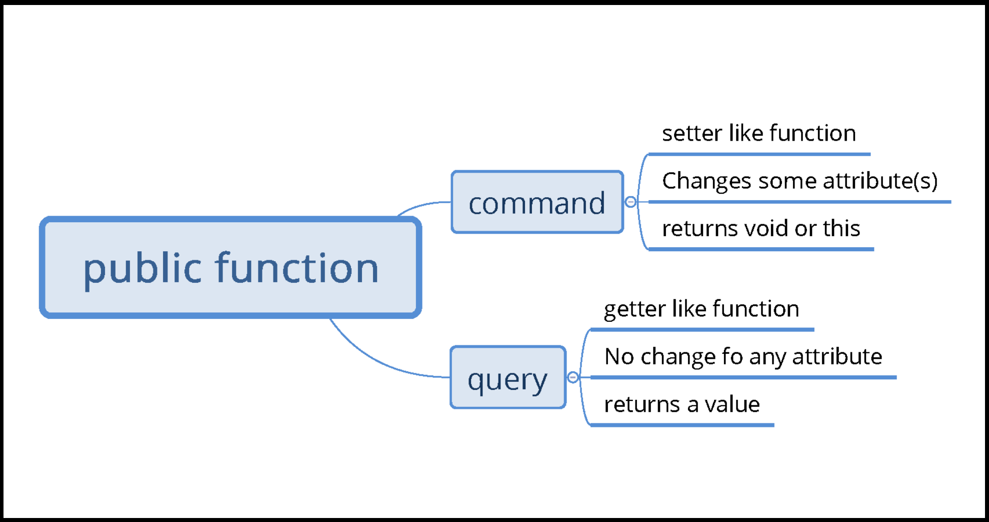
- Keep the command and queries segregated
  - Accessors, mutators, and predicates should be named for their value and prefixed with get, set, and is according to the standard.

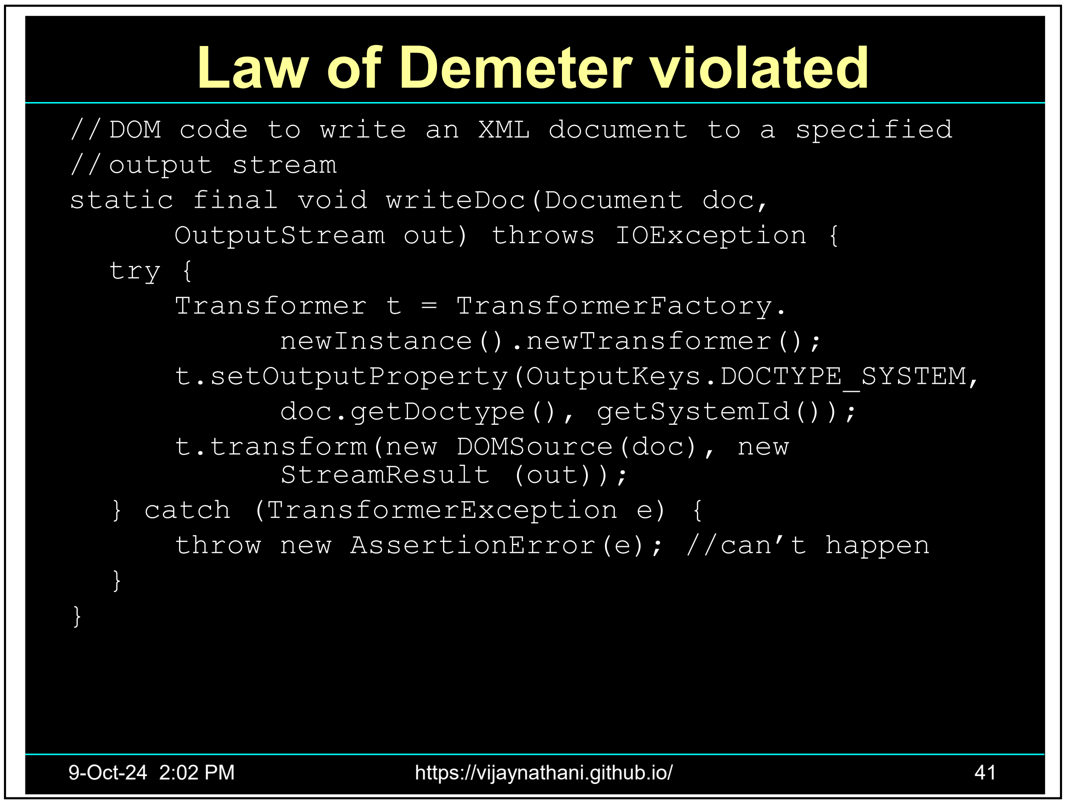
```
String name =
  employee.getName();
customer.setName("mike");
if (paycheck.isPosted())...
```

Exception: DSL.

Commands : return void.

Command vs Query





Law of Demeter violated

```
//DOM code to write an XML document to a specified
//output stream
static final void writeDoc(Document doc,
      OutputStream out) throws IOException {
  try {
      Transformer t = TransformerFactory.
            newInstance().newTransformer();
      t.setOutputProperty(OutputKeys.DOCTYPE_SYSTEM,
            doc.getDoctype(), getSystemId());
      t.transform(new DOMSource(doc), new
            StreamResult (out));
  } catch (TransformerException e) {
      throw new AssertionError(e); //can't happen
  }
}
```

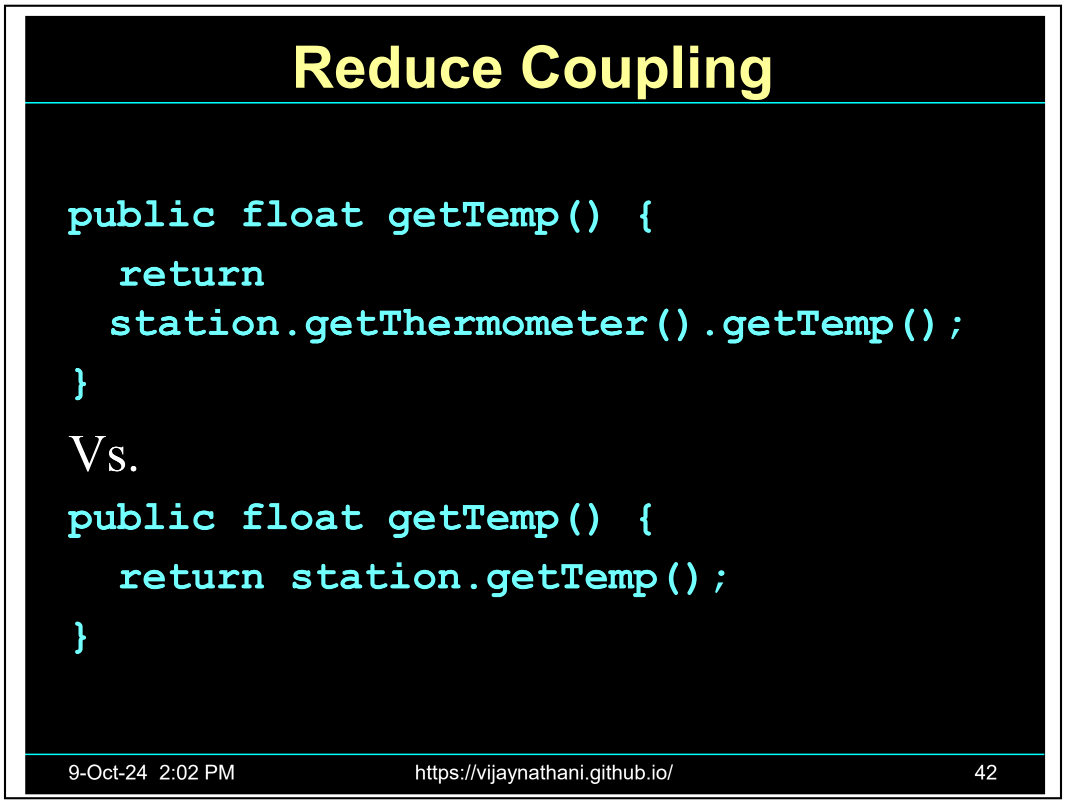
9-Oct-24 2:02 PM https://vijaynathani.github.io/ 41

Principle of least knowledge or Law of Demeter:

Each unit should only talk to its friends; Don't talk to strangers.

Don’t make the client do anything, that the Module could do

- Reduce the need for boilerplate code
- Generally done via cut-and-paste
- Ugly, Annoying and error-prone

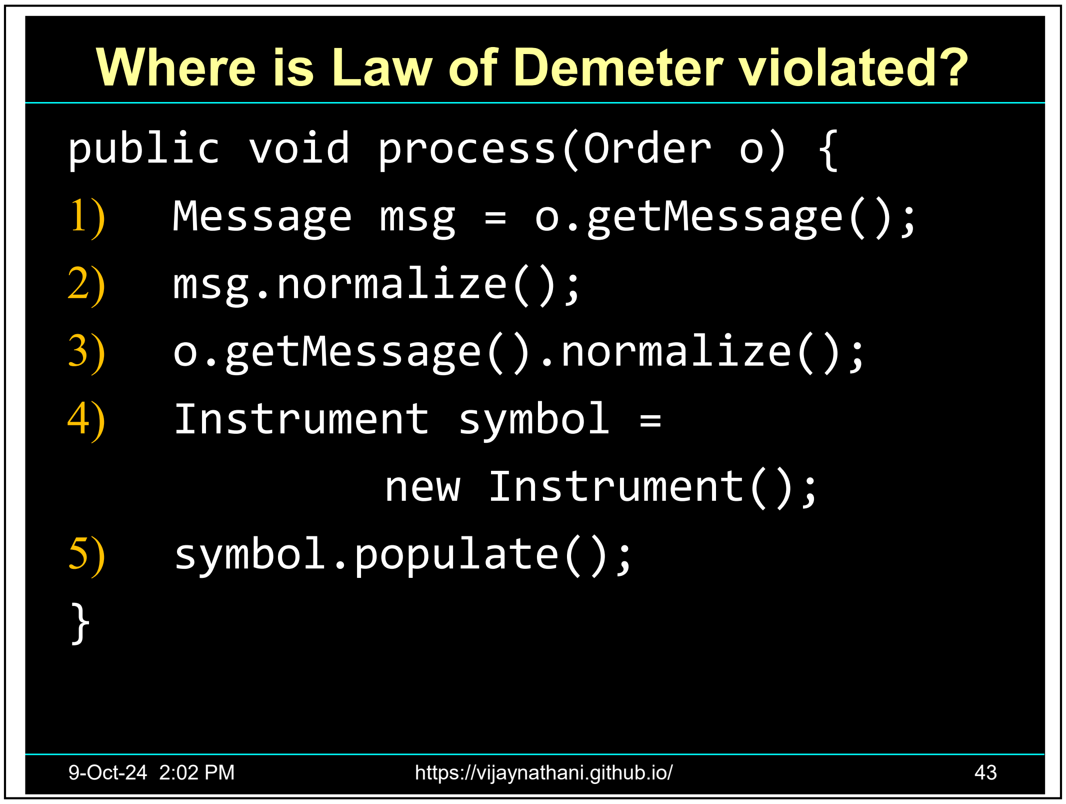


Reduce Coupling

```
public float getTemp() {
   return
  station.getThermometer().getTemp();
 }
```

Vs.

```
public float getTemp() {
   return station.getTemp();
 }
```

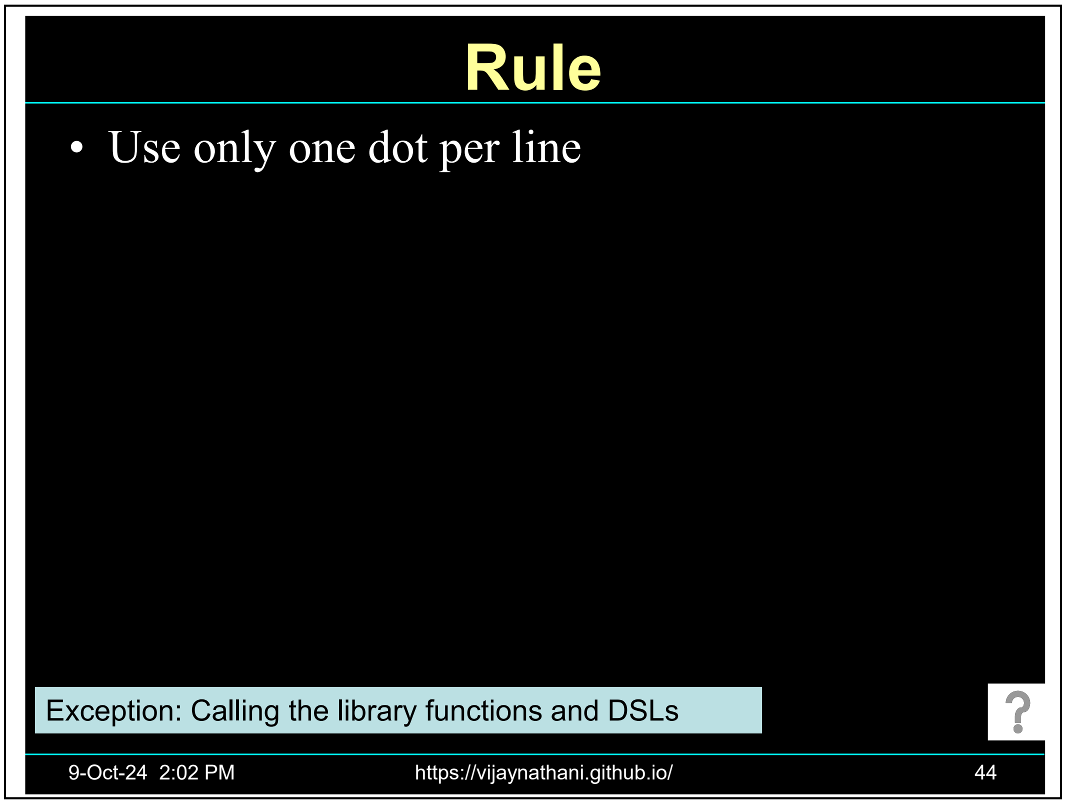


Where is Law of Demeter violated?

```
public void process(Order o) {
1)   Message msg = o.getMessage();
2)   msg.normalize();
3)   o.getMessage().normalize();
4)   Instrument symbol =
                new Instrument();
5)   symbol.populate();
}
```

9-Oct-24 2:02 PM https://vijaynathani.github.io/ 43

Answer) lines 2 and 3



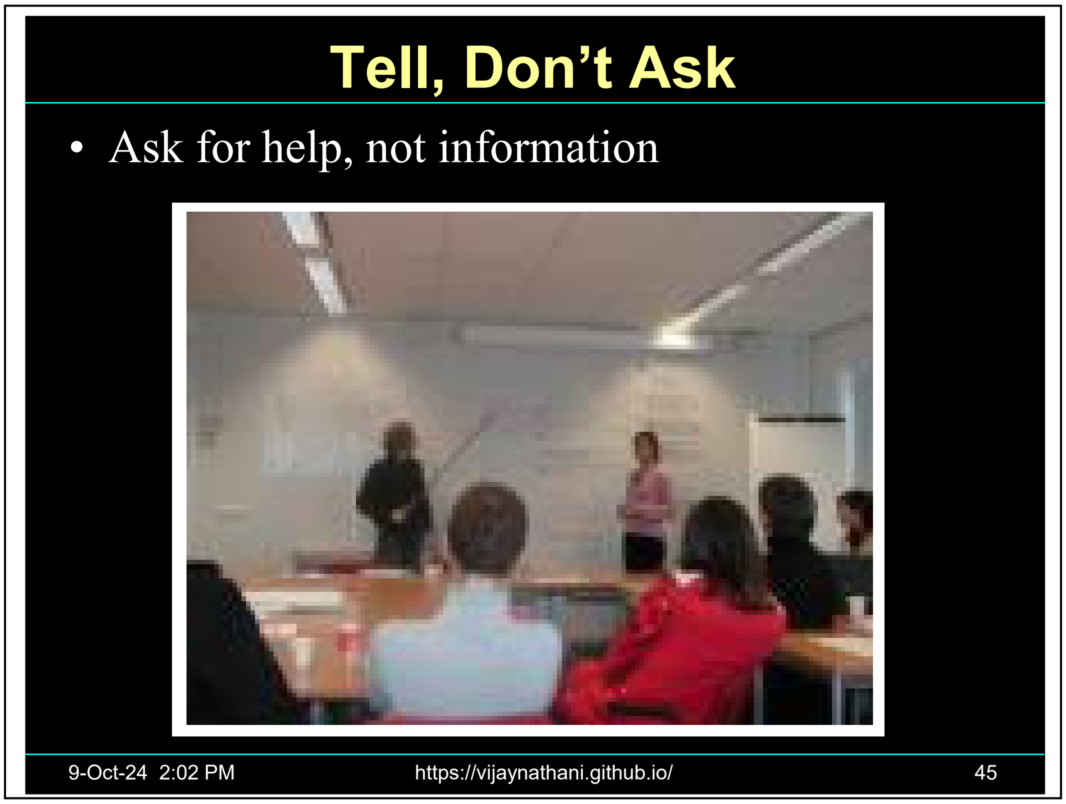
Rule

- Use only one dot per line

Exception: Calling the library functions and DSLs

?

Q26Demeter.



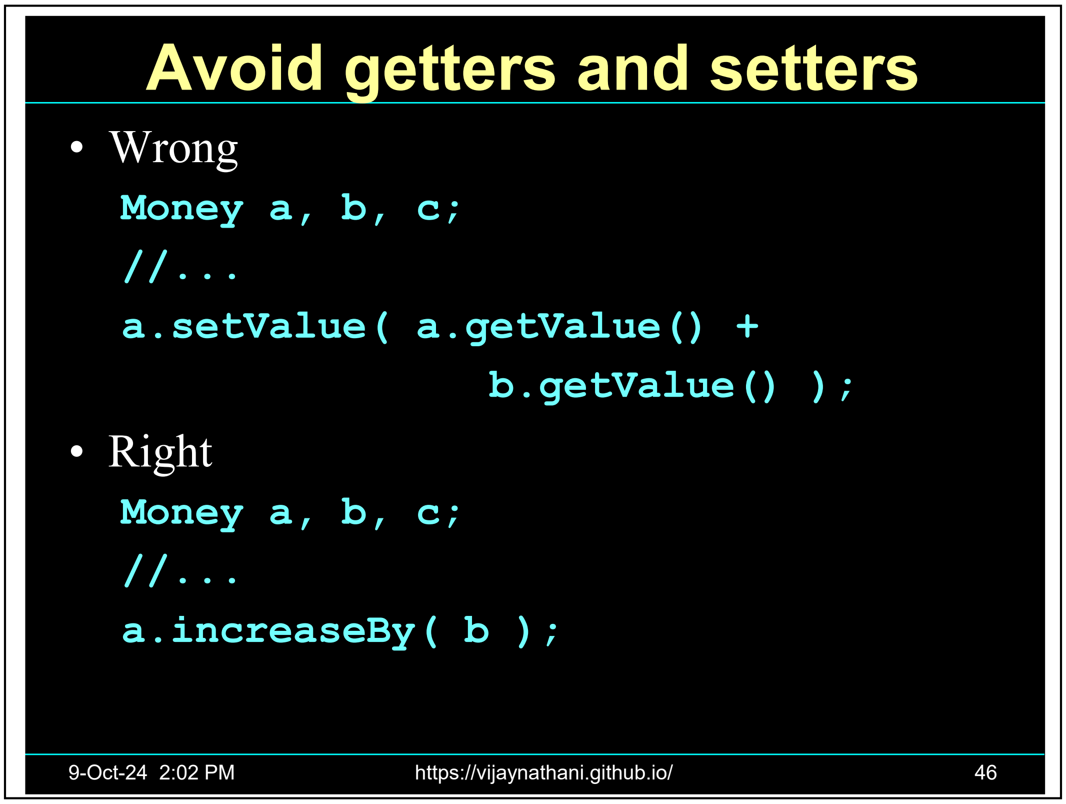
Tell, Don’t Ask

- Ask for help, not information

9-Oct-24 2:02 PM https://vijaynathani.github.io/ 45

Never ask an object for information that you need to do something; rather, ask the object that has the information to do the work for you.

In other words: Don’t use any getters/setters/properties.



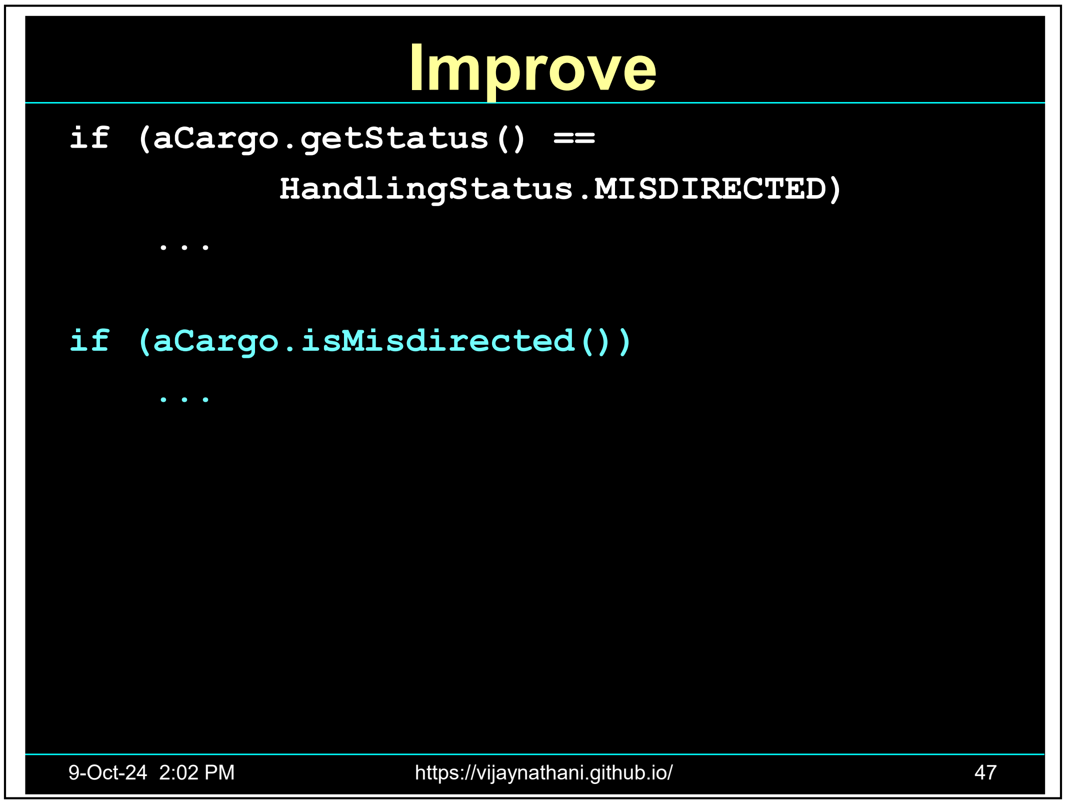
Avoid getters and setters

- Wrong

```
Money a, b, c;
//...
a.setValue( a.getValue() +
               b.getValue() );
```

- Right

```
Money a, b, c;
//...
a.increaseBy( b );
```

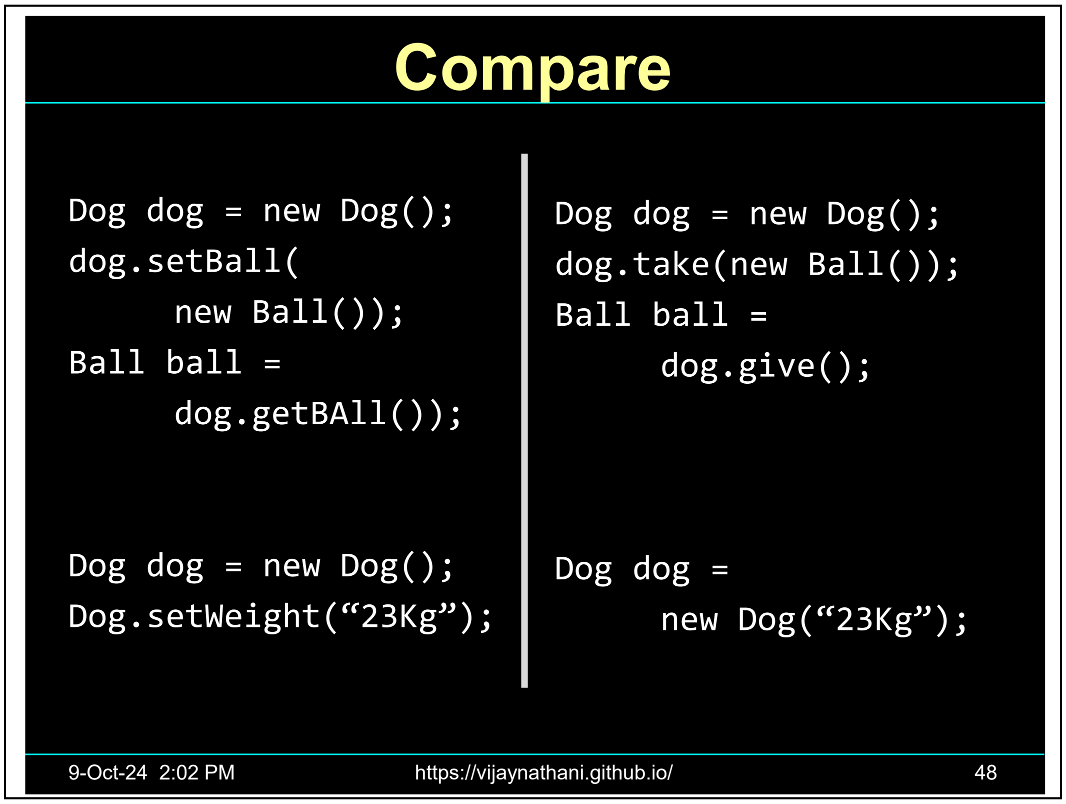


Improve

```
if (aCargo.getStatus() ==
          HandlingStatus.MISDIRECTED)
    ...

if (aCargo.isMisdirected())
    ...
```

9-Oct-24 2:02 PM https://vijaynathani.github.io/ 47



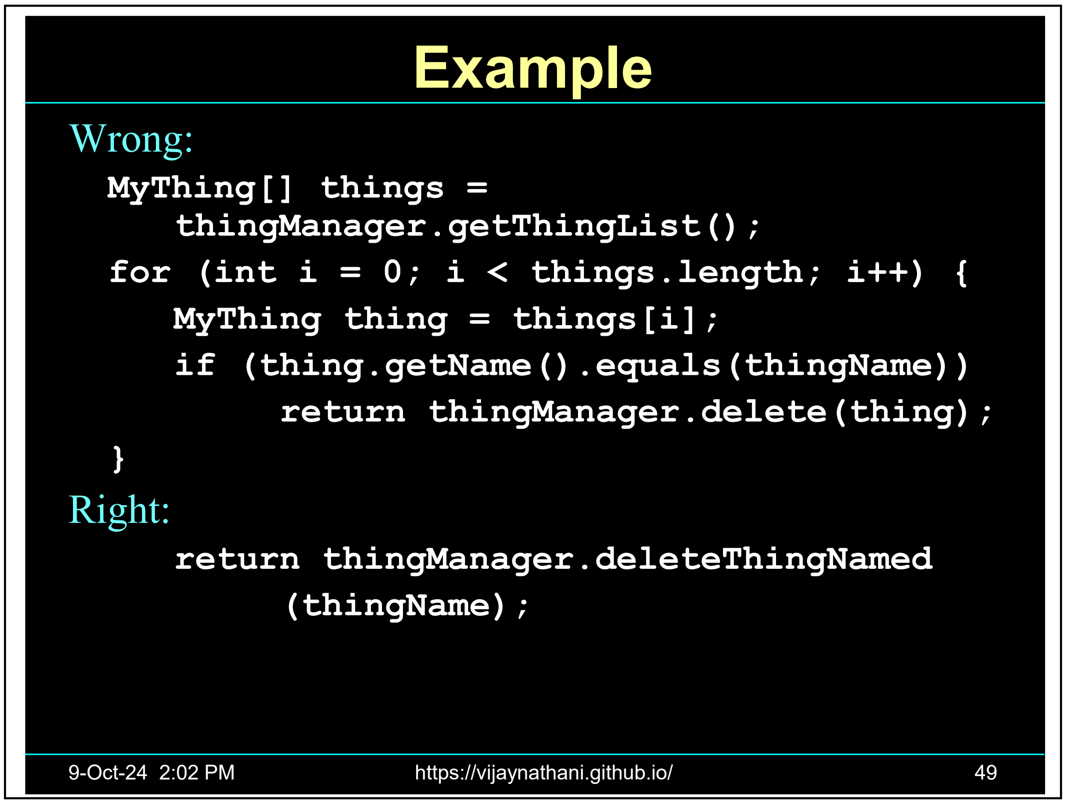
Compare

```
Dog dog = new Dog();
dog.setBall(
      new Ball());
Ball ball =
      dog.getBAll());
```

```
Dog dog = new Dog();
dog.take(new Ball());
Ball ball =
      dog.give();
```

```
Dog dog = new Dog();
Dog.setWeight(“23Kg”);
```

```
Dog dog =
      new Dog(“23Kg”);
```



Example

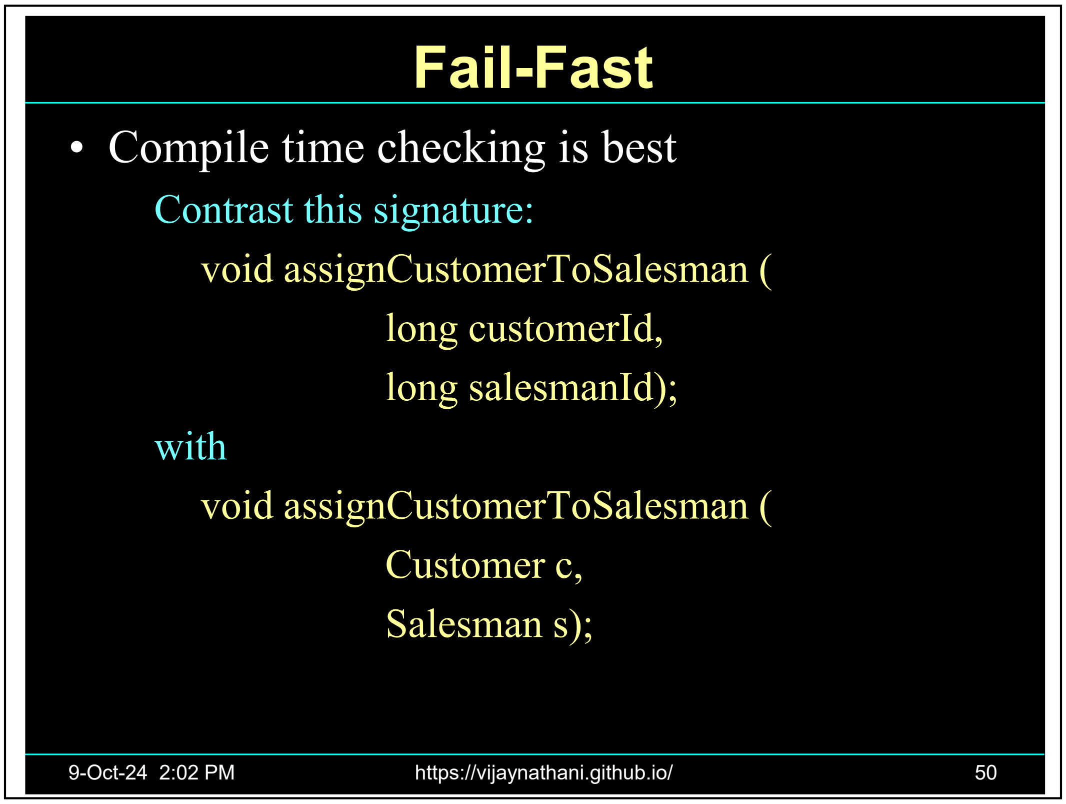
Wrong:

```
MyThing[] things =
   thingManager.getThingList();
for (int i = 0; i < things.length; i++) {
   MyThing thing = things[i];
   if (thing.getName().equals(thingName))
        return thingManager.delete(thing);
}
```

Right:

```
   return thingManager.deleteThingNamed
        (thingName);
```

9-Oct-24 2:02 PM https://vijaynathani.github.io/ 49



Fail-Fast

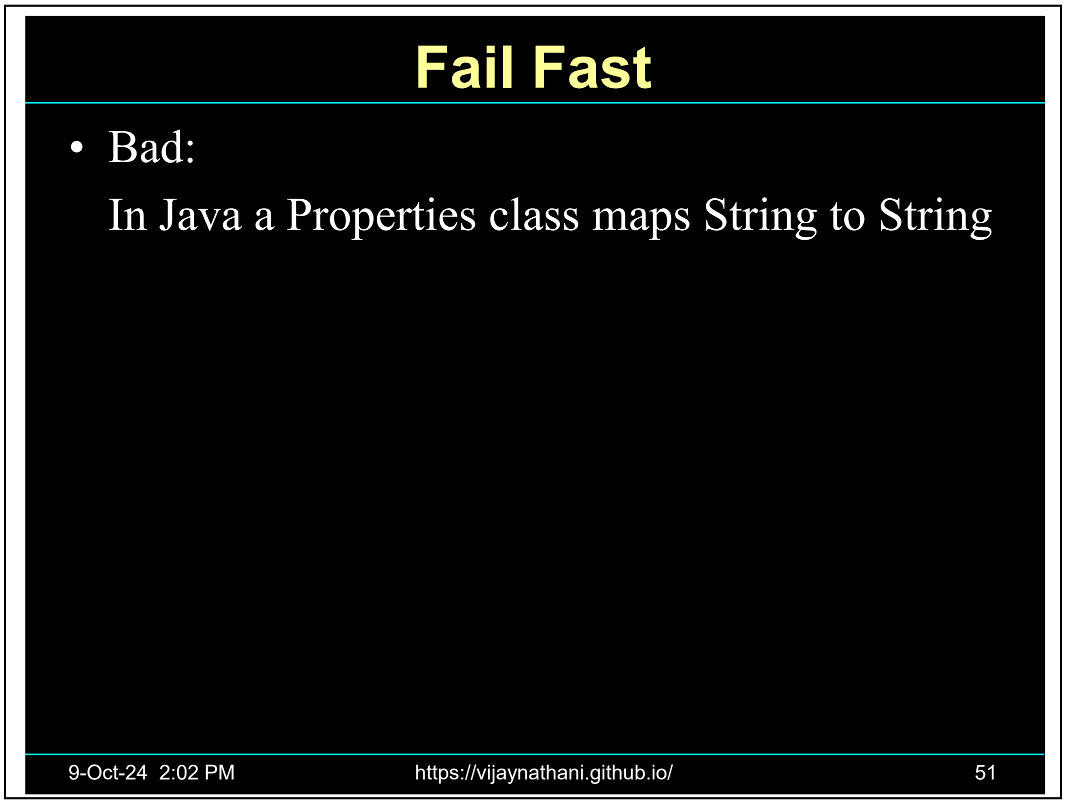
- Compile time checking is best

  Contrast this signature:

```
void assignCustomerToSalesman (
                long customerId,
                long salesmanId);
```

  with

```
void assignCustomerToSalesman (
                Customer c,
                Salesman s);
```

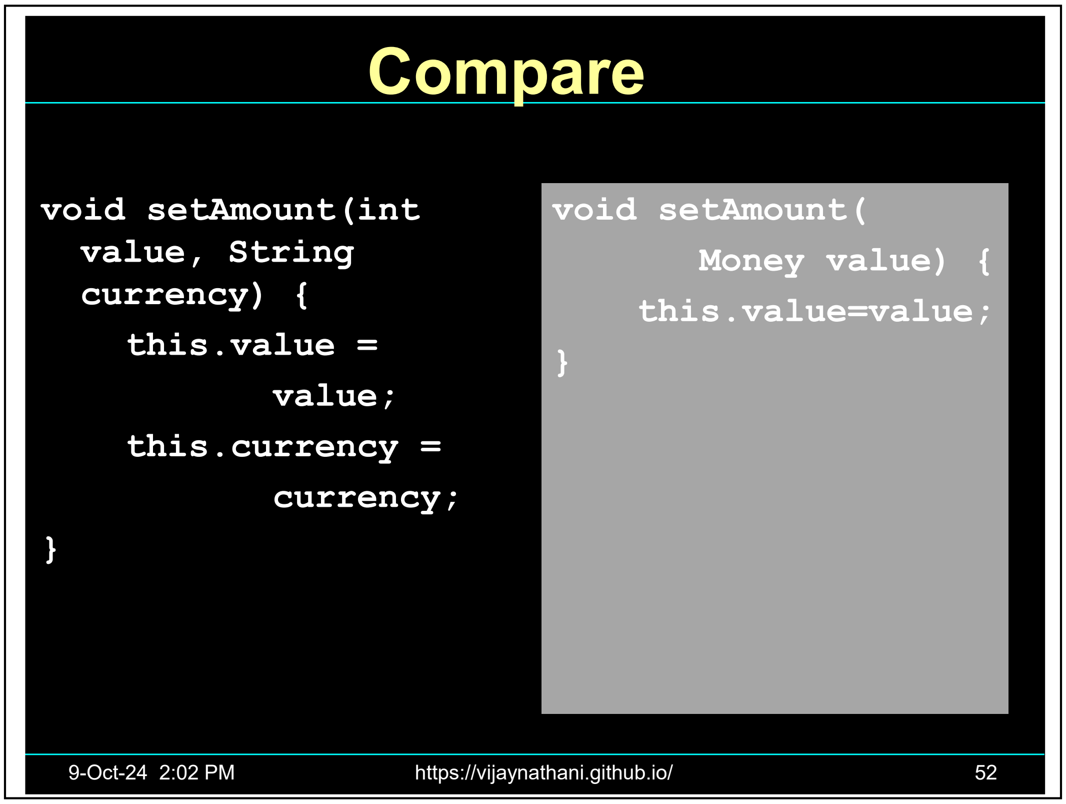

Fail Fast

- Bad:

  In Java a Properties class maps String to String

put method does not make this check

save method does this check



Compare

```
void setAmount(int
  value, String
  currency) {
    this.value =
           value;
    this.currency =
           currency;
}
```

```
void setAmount(
       Money value) {
    this.value=value;
}
```

9-Oct-24 2:02 PM https://vijaynathani.github.io/ 52

The code on the right

•Is flexible because any runner can be used

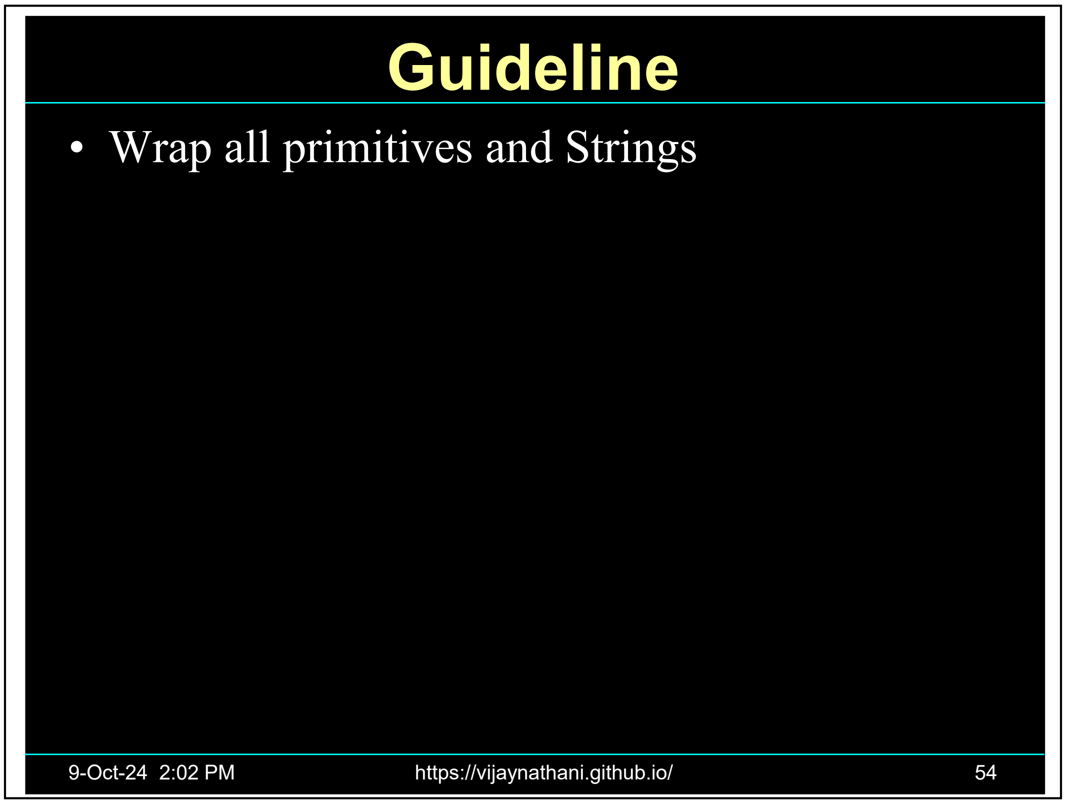
•Less prone to error, because the caller does not have to worry about exceptions

Improve

```
setOuterBounds ( x, y,
          width, height);
setInnerBounds ( x+2, y+2,
          width-4, height-4);
```

- Solution: Use Parameter Object

```
setOuterBounds (bounds);
setInnerBounds (bounds.expand
                  (-2));
```



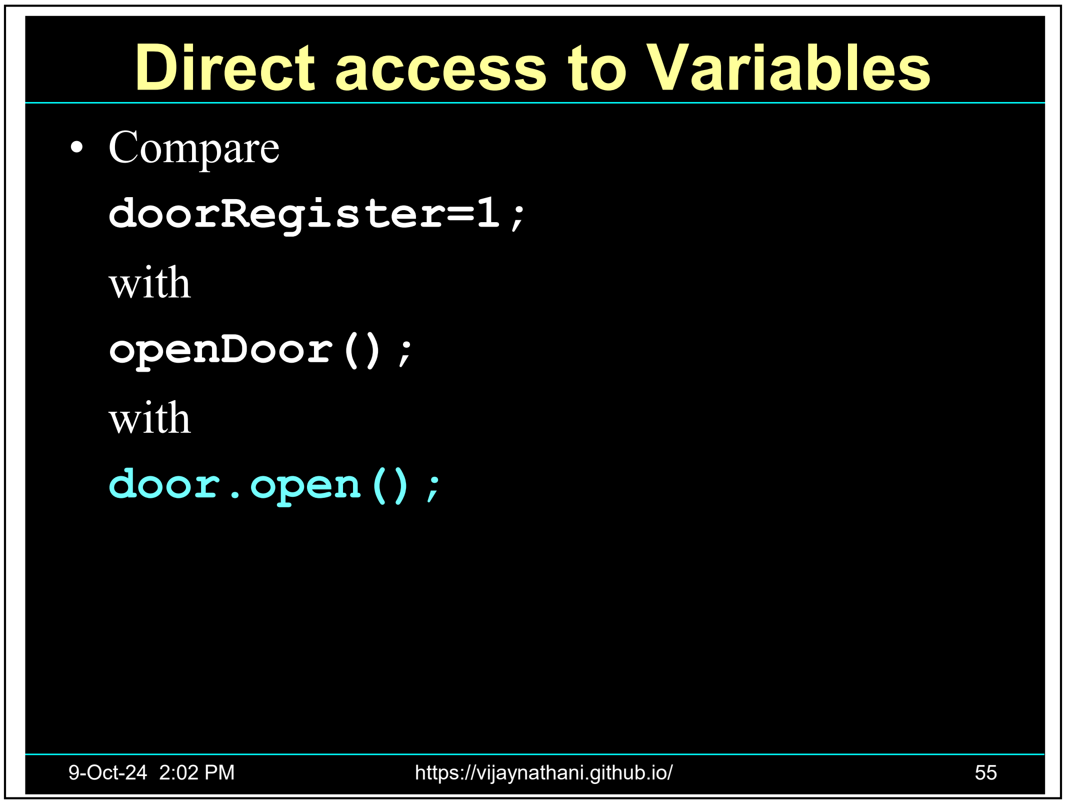
Guideline

- Wrap all primitives and Strings

9-Oct-24 2:02 PM https://vijaynathani.github.io/ 54

An int on its own is just a scalar with no meaning. When a method takes an int as a parameter, the method name needs to do all the work of expressing the intent. If the same method takes an hour as a parameter, it's much easier to see what's happening. Small objects like this can make programs more maintainable, since it isn't possible to pass a year to a method that takes an hour parameter. With a primitive variable, the compiler can't help you write semantically correct programs. With an object, even a small one, you are giving both the compiler and the programmer additional information about what the value is and why it is being used.

Small objects such as hour or money also give you an obvious place to put behavior that otherwise would have been littered around other classes. This becomes especially true when you apply the rule relating to getters and setters and only the small object can access the value.



Direct access to Variables

- Compare

  `doorRegister=1;`

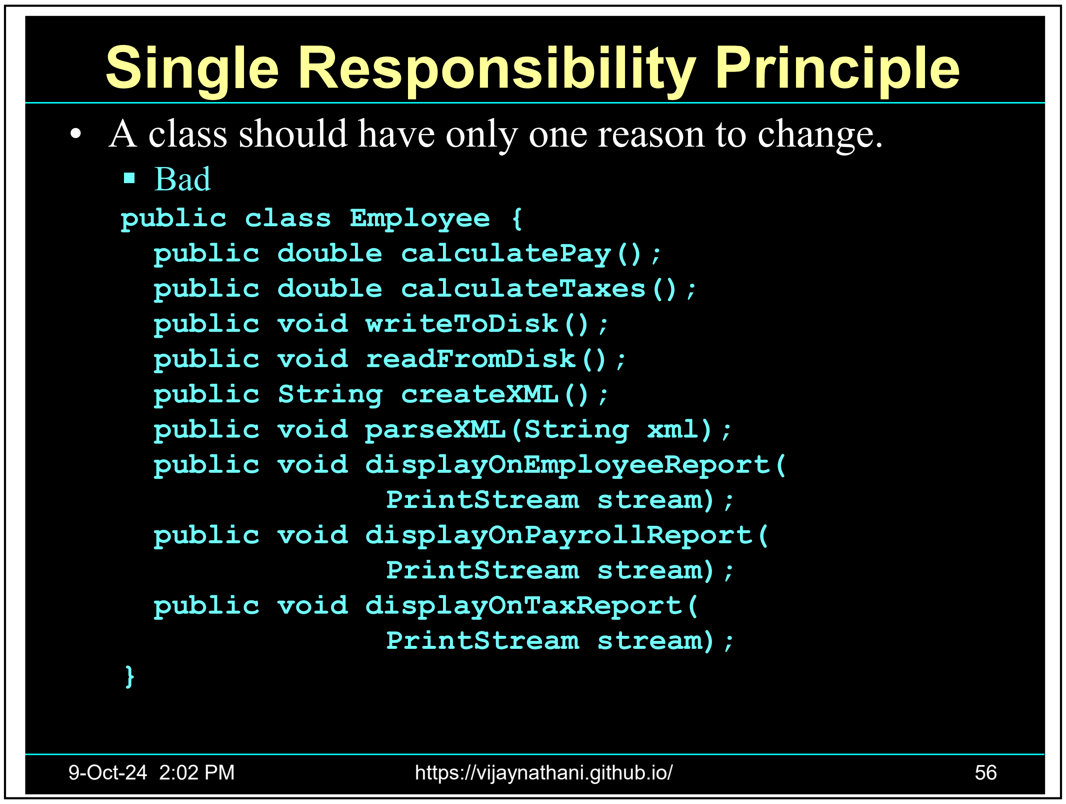
  with

  `openDoor();`

  with

  `door.open();`

The last one is the best. It uses objects and functions.



Single Responsibility Principle

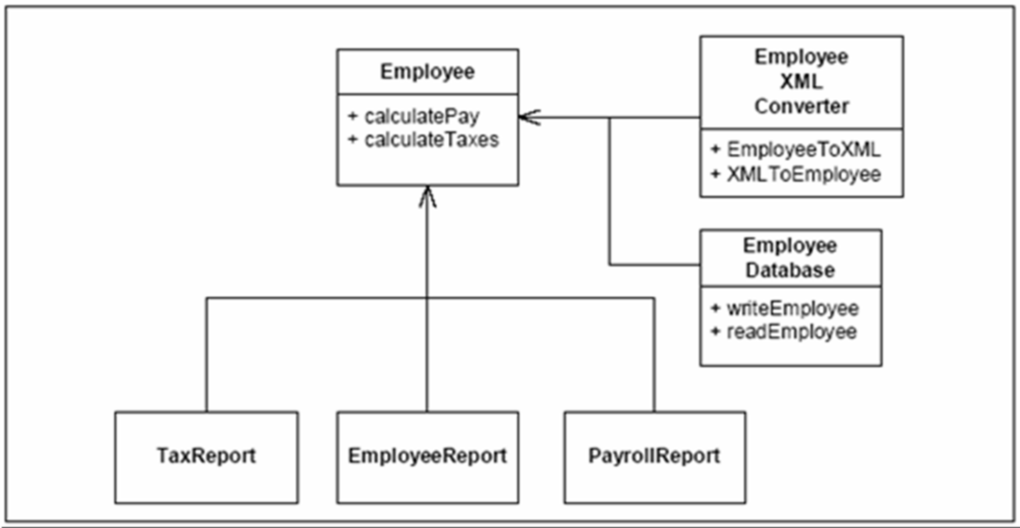
- A class should have only one reason to change.
  - Bad

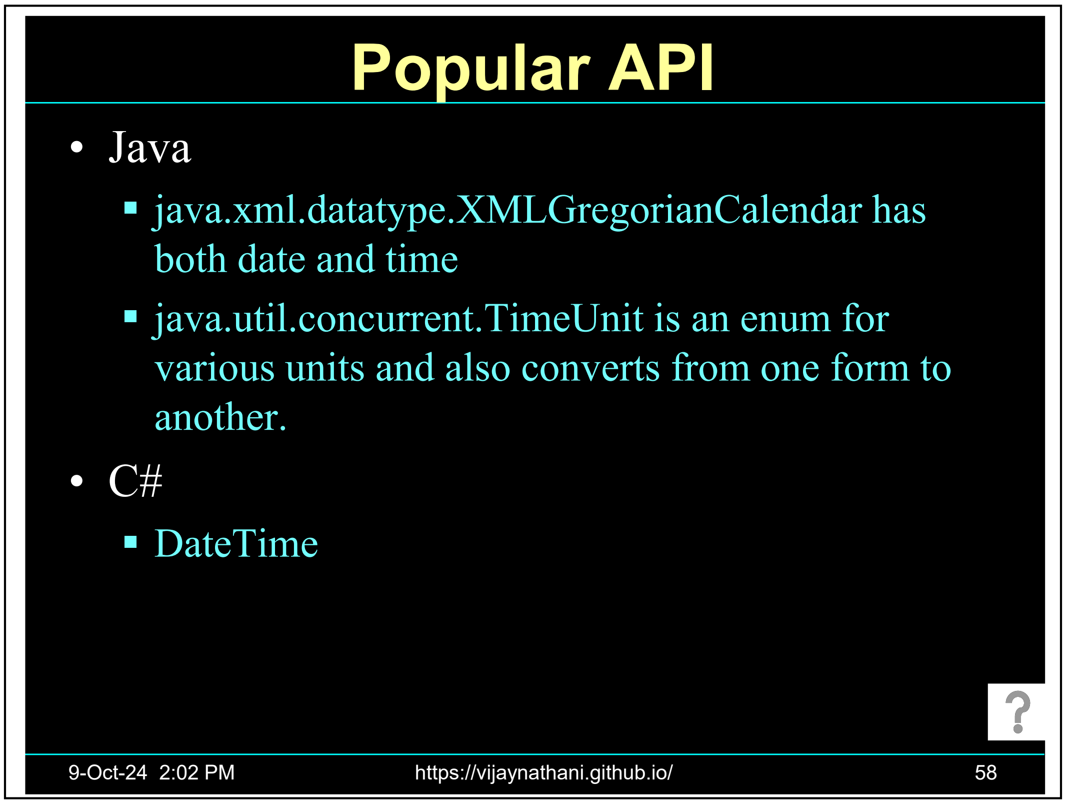
```
public class Employee {
  public double calculatePay();
  public double calculateTaxes();
  public void writeToDisk();
  public void readFromDisk();
  public String createXML();
  public void parseXML(String xml);
  public void displayOnEmployeeReport(
               PrintStream stream);
  public void displayOnPayrollReport(
               PrintStream stream);
  public void displayOnTaxReport(
               PrintStream stream);
}
```

9-Oct-24 2:02 PM https://vijaynathani.github.io/ 56

An item such as a class should just have one responsibility and solve that responsibility well. If a class is responsible both for presentation and data access, that's a good example of a class breaking SRP.

SRP implemented





Popular API

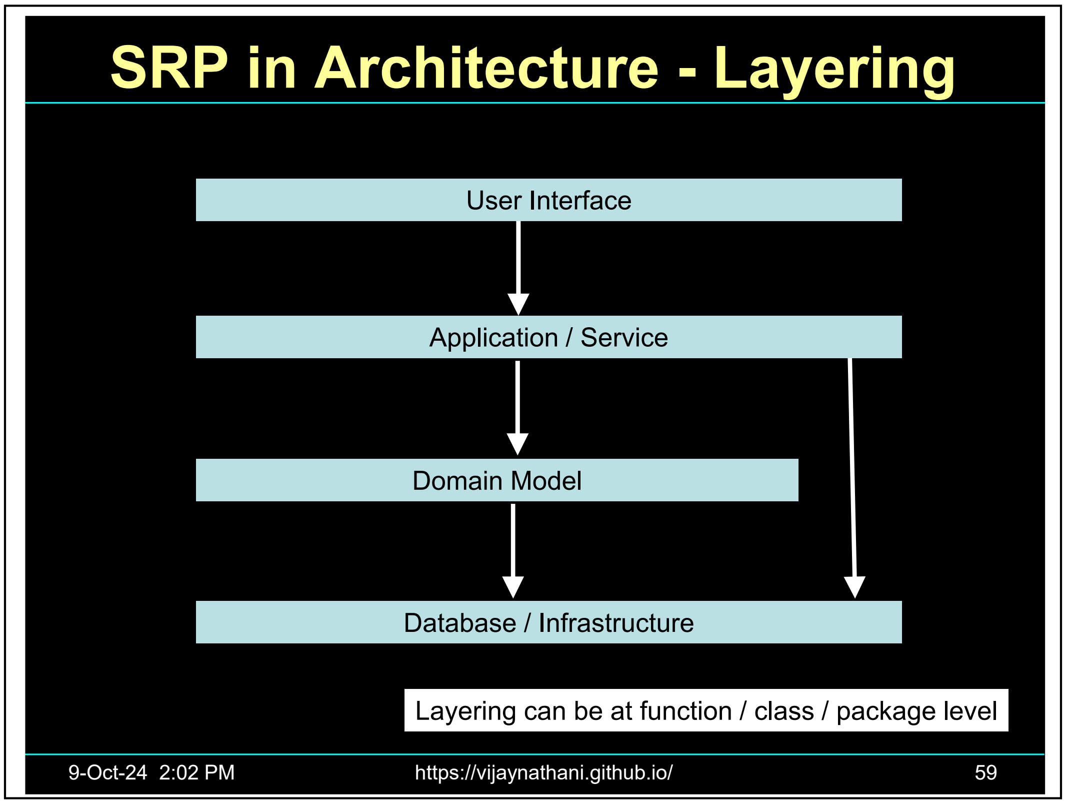
- Java
  - java.xml.datatype.XMLGregorianCalendar has both date and time
  - java.util.concurrent.TimeUnit is an enum for various units and also converts from one form to another.
- C#
  - DateTime

9-Oct-24 2:02 PM https://vijaynathani.github.io/ 58

Q50 – Account

Q51 – Department

Q55 – processReport1



SRP in Architecture - Layering

Layering can be at function / class / package level

9-Oct-24 2:02 PM https://vijaynathani.github.io/ 59

Service / Application layer co-ordinates tasks and delegates work to the domain. In an rich domain model, Service layer should be as thin as possible.

Interactions with the legacy systems is at Infrastructure level.

Usually the service layer

1. Gets the Database objects by interacting directly with Database layer and then
2. The domain model is executed in the objects.

In Java EE 1.4 or less,

Session beans are at service layer

Entity beans are at Database layer.

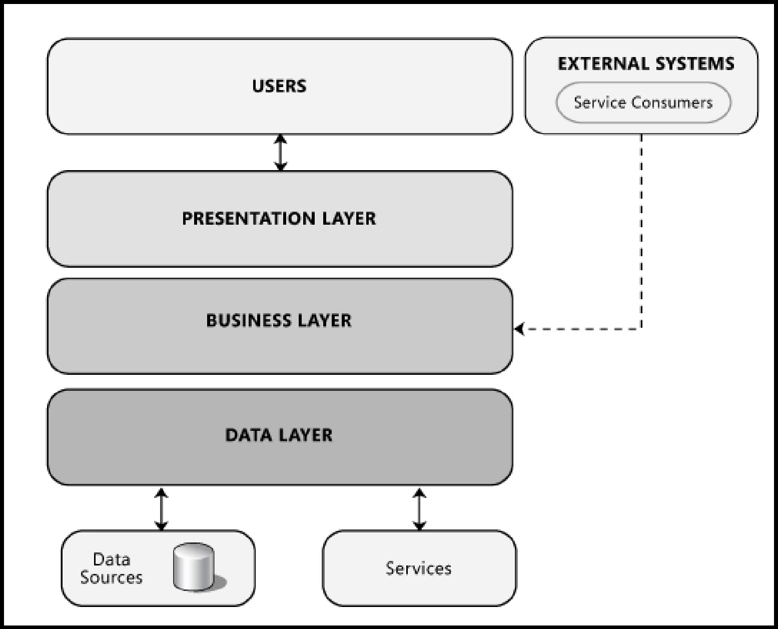
If business logic code is kept in Session beans, then it will become transaction script.

If business logic code is kept in Entity beans, then it will become Table Module

Table module is popular in .NET. Transaction script was popular with VB6.

One of the hardest parts of working with domain logic seems to be that people often find it difficult to recognize what is domain logic and what is other forms of logic. An informal test I like is to imagine adding a radically different layer to an application, such as a command-line interface to a Web application. If there's any functionality you have to duplicate in order to do this, that's a sign of where domain logic has leaked into the presentation. Similarly, do you have to duplicate logic to replace a relational database with an XML file?

Layering



Layers are logical separation. Tiers are physical separation. It is possible to run all the four layers in the same tier.

A rich domain model needs Transaction management, Security enforcement and Remote management.

===================================

Low coupling between layers. High cohesion between them.

User interface modules should not contain business logic.

No circular references between layers.

Business layer uses abstraction of technological services.

Layers should be testable individually.

Layers are a logical abstract and not necessarily physical.

Layers should be shy about their internals.