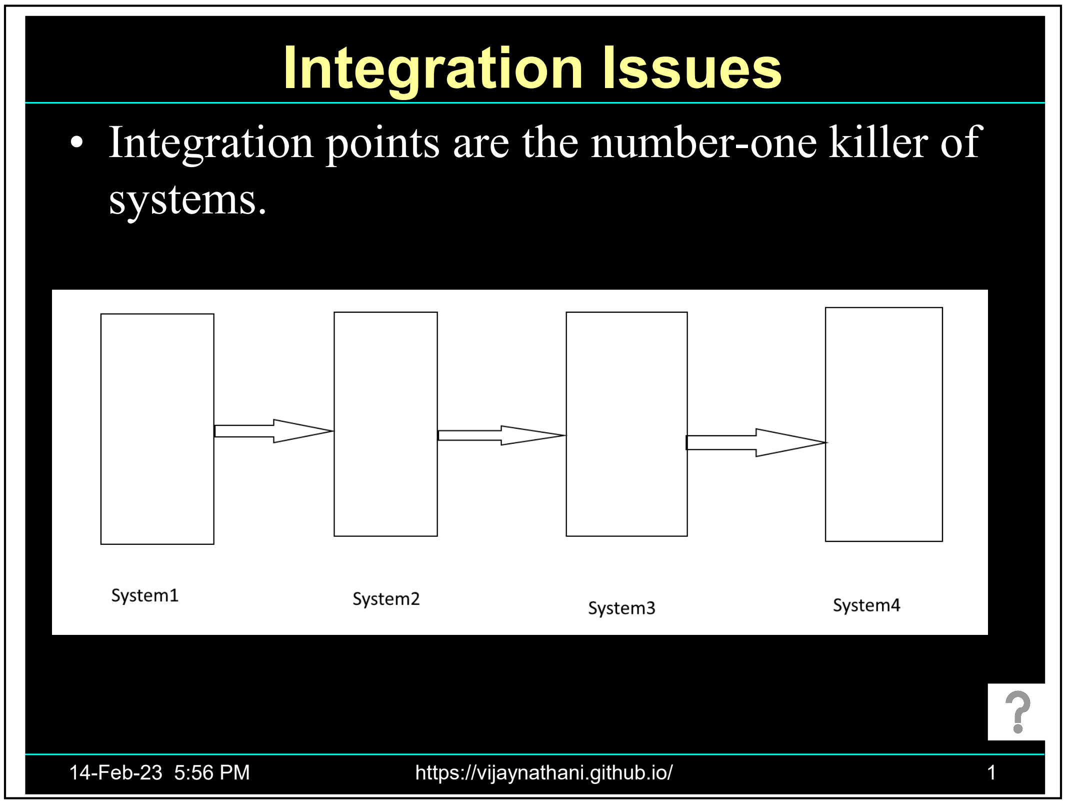
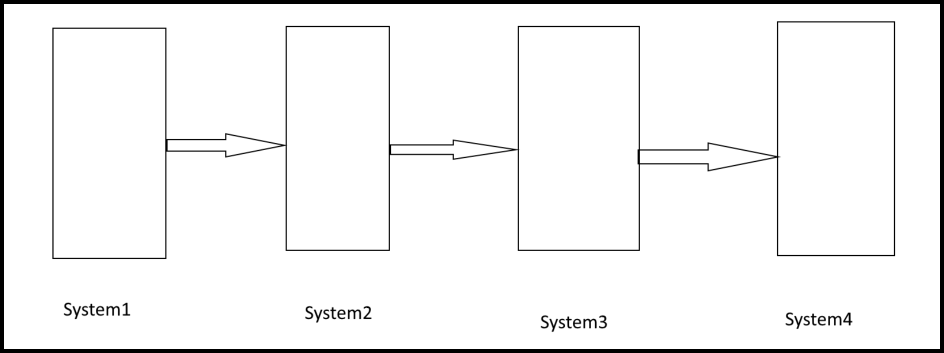


Integration Issues

- Integration points are the number-one killer of systems.



Circuit breaker code in is SCircuitBreaker.

Connection timeouts for TCP/IP is in minutes, while creating a new connection. So the thread remains block. All threads in a pool can get blocked.

That is why I advocate supplementing internal monitors (such as log file scraping, process monitoring, and port monitoring) with external monitoring. A mock client somewhere (not in the same data center) can run synthetic transactions on a regular basis. That client experiences the same view of the system that real users experience. When that client cannot process the synthetic transactions, then there is a problem, whether or not the server process is running.

Blocked Threads are a major source of gradual slowdown and hung server. Use timeouts for sockets and library calls that block e.g. in java.util.concurrent.*

Use Circuit breaker e.g. After load crosses a certain limit, refuse more connections. E.g. If your website cannot handle high volume, then detect this high volume surge and give the users a polite message to come back later.

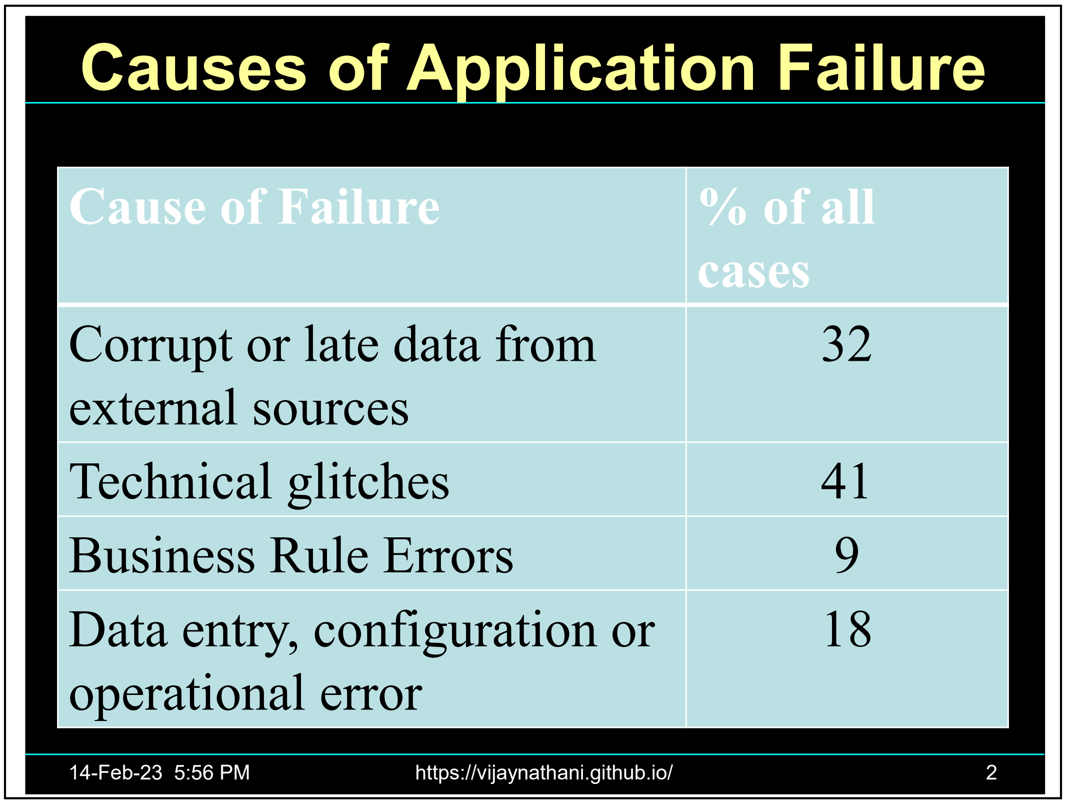
Integration Points:

Every single one of those feeds presents a stability risk.

Every socket, process, pipe, or remote procedure call can and will hang.

Even database calls can hang, in ways obvious and subtle.

Every feed into the system can hang it, crash it, or generate other impulses at the worst possible time.



Causes of Application Failure

Cause of Failure	% of all cases
Corrupt or late data from external sources	32
Technical glitches	41
Business Rule Errors	9
Data entry, configuration or operational error	18

14-Feb-23 5:56 PM https://vijaynathani.github.io/ 2

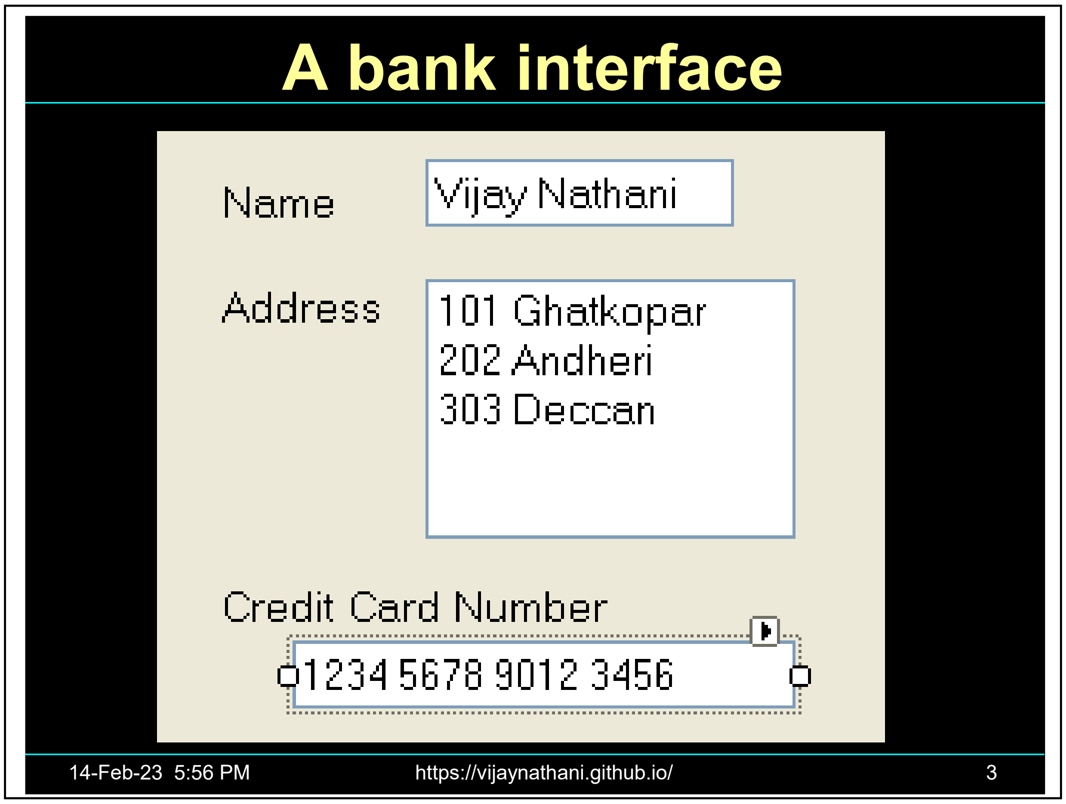
Source: **High-Assurance Design: Architecting Secure and Reliable Enterprise Applications** By Clifford J. Berg

Most failures resulted from various and sundry technical "glitches” situations where the software designer might have said, "That is unlikely to happen," and the support staff later said, "We'll never be able to reproduce that." This category also includes those errors that cannot be explained by the technical staff. E.g. Virus, SQL Injection, Denial of Service, Sniffing, Man in the middle, etc.

Developers and Testers tend to concentrate on “Business Rule Errors”

Operational error: An intern in hospital moves about many departments in his/her few months of work at hospital. By the end of these few months, he/she has more rights on the system than many permanent doctors. People call when they don’t have access rights. Nobody calls when access rights have to be removed.

A bank interface



Name	Vijay Nathani
Address	101 Ghatkopar 202 Andheri 303 Deccan
Credit Card Number	1234 5678 9012 3456

14-Feb-23 5:56 PM https://vijaynathani.github.io/ 3

A banking representative asks:

1) Name
2) Give me your address
3) Give me last four digits of your credit card number.

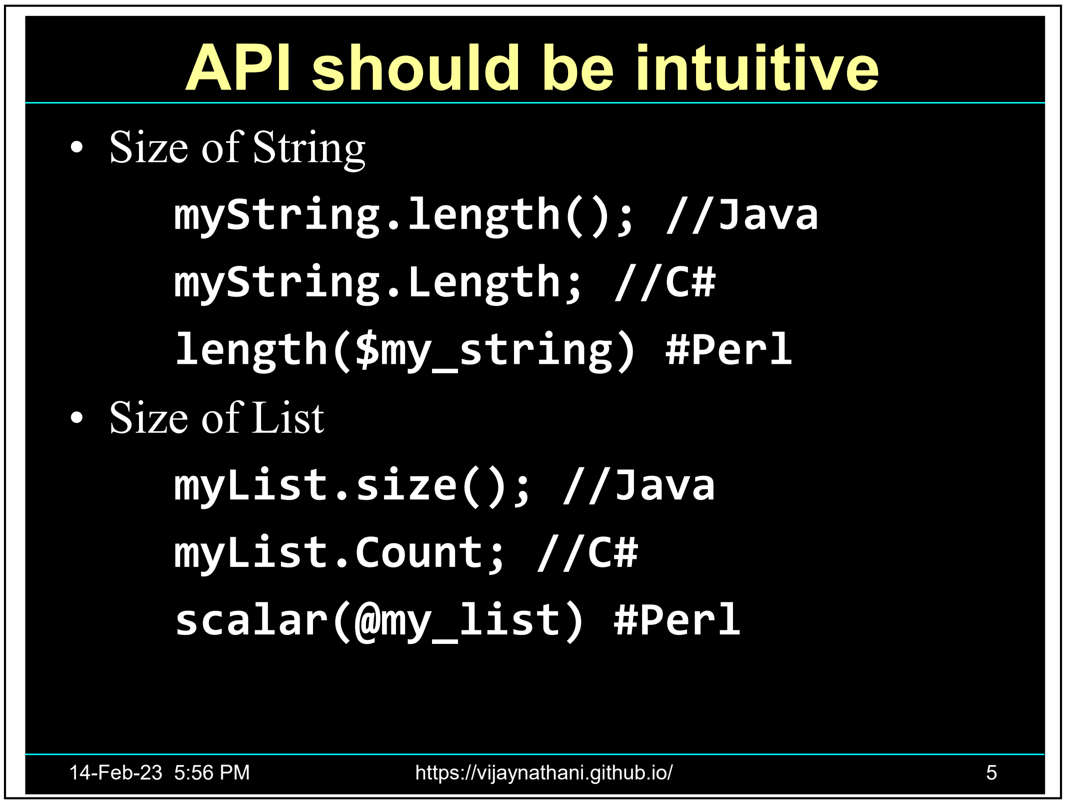
The person is getting all the data. He is transient, located in a remote area and probably lowly paid. This is high risk.

In a better design, he should

1) Enter the last four digits and the system should say match or not.
2) He should enter at least a part of address e.g. house number.

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

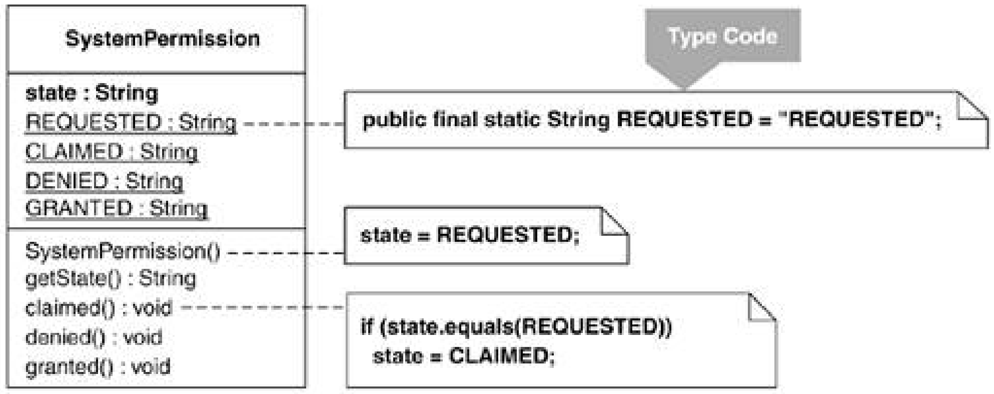
```
    myList.size(); //Java
    myList.Count; //C#
    scalar(@my_list) #Perl
```

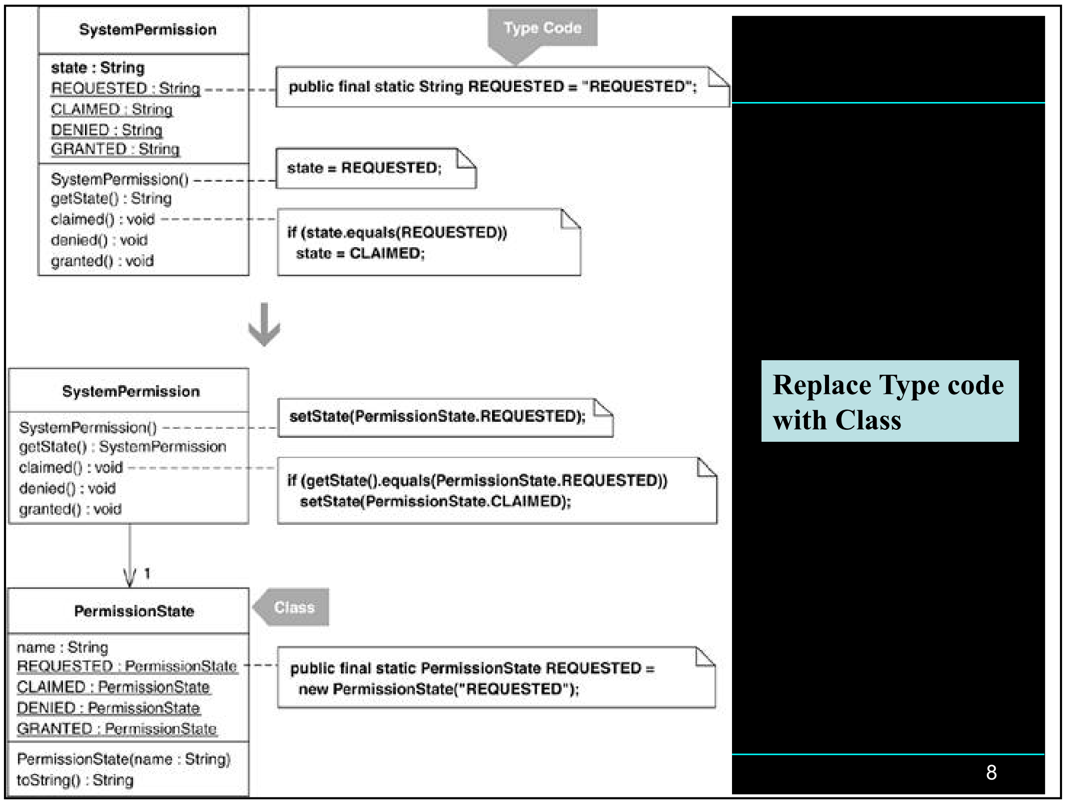
14-Feb-23 5:56 PM https://vijaynathani.github.io/ 5

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;
```

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

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

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

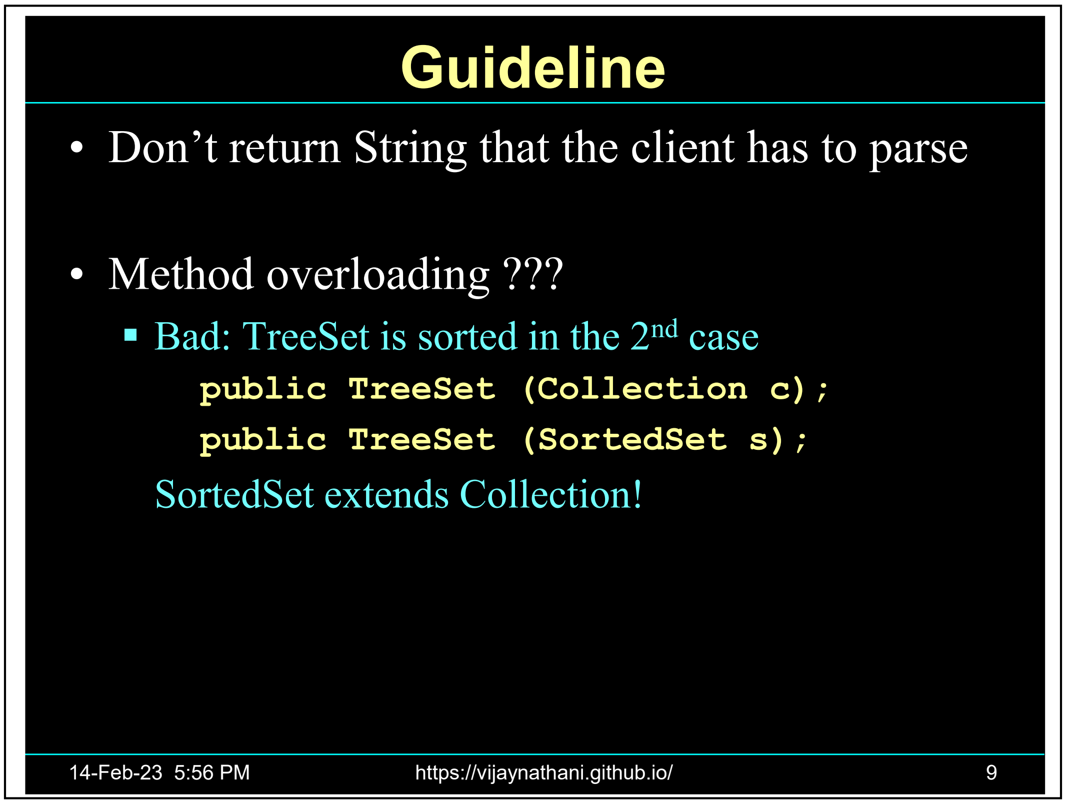
Replace Type code with Class

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!

Return of sting with multiple values is bad because

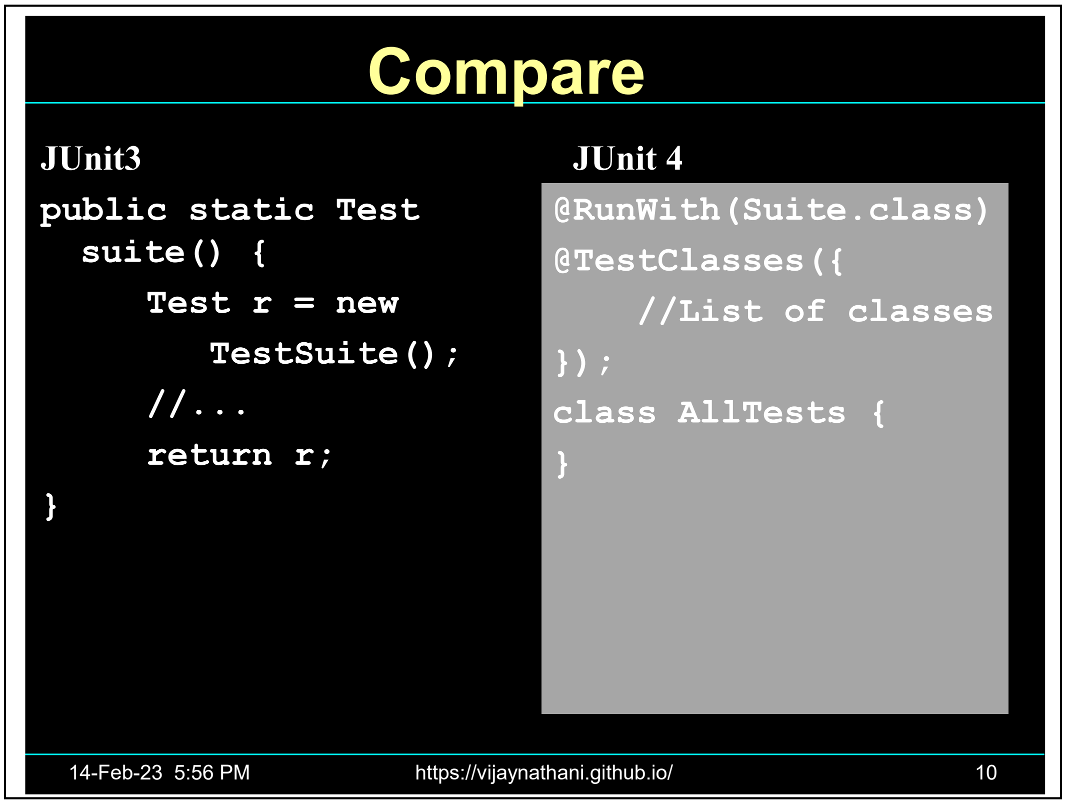
Future modifications will become difficult

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 {
}
```

14-Feb-23 5:56 PM https://vijaynathani.github.io/ 10

The code on the right

•Is flexible because any runner can be used

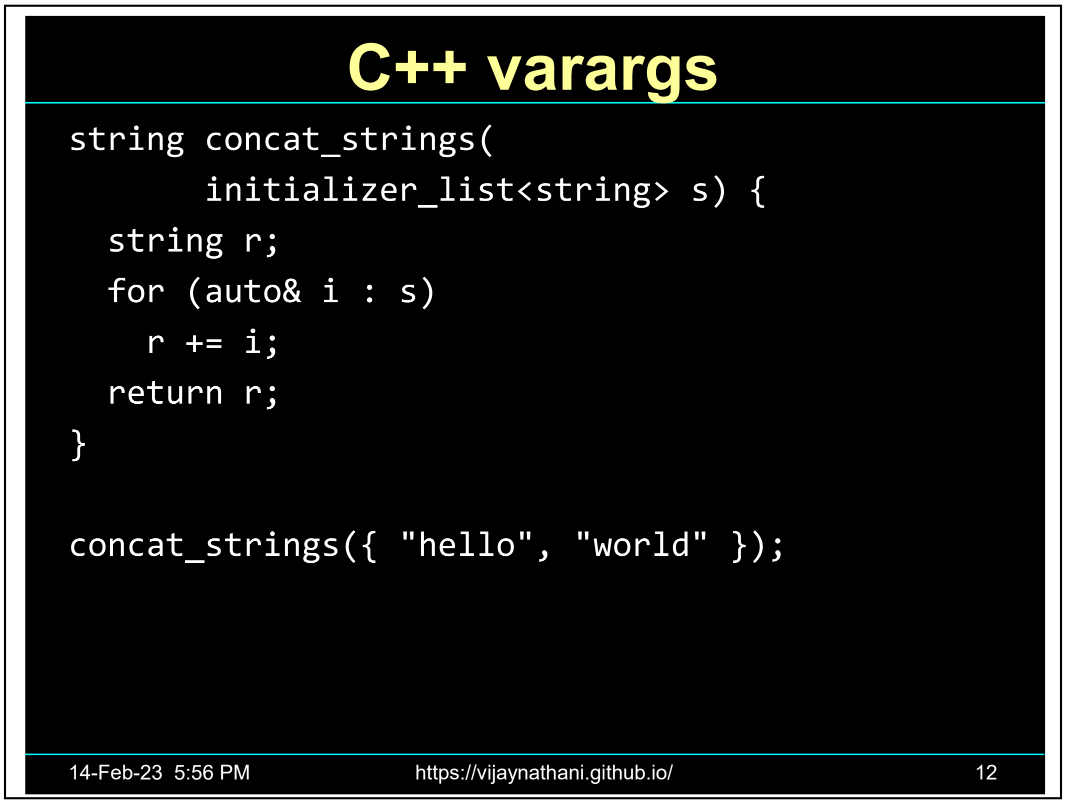
•Less prone to error, because the caller does not have to worry about exceptions

Improve

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

- Use varargs

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



C++ varargs

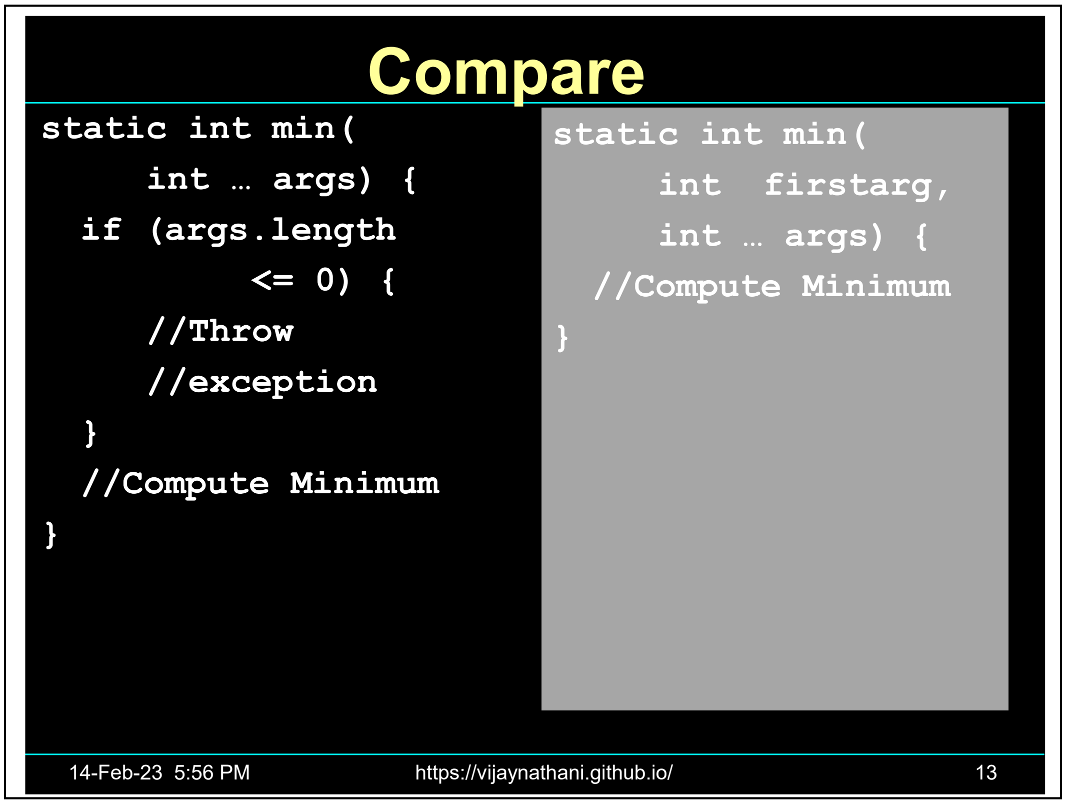
```
string concat_strings(
       initializer_list<string> s) {
  string r;
  for (auto& i : s)
    r += i;
  return r;
}

concat_strings({ "hello", "world" });
```

14-Feb-23 5:56 PM https://vijaynathani.github.io/ 12

Initializer_list is present from C++11 onwards.

In old versions of C++, vector can be used instead. Vector is a more heavy-weight as compared to initializer_list.



Compare

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

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

14-Feb-23 5:56 PM https://vijaynathani.github.io/ 13

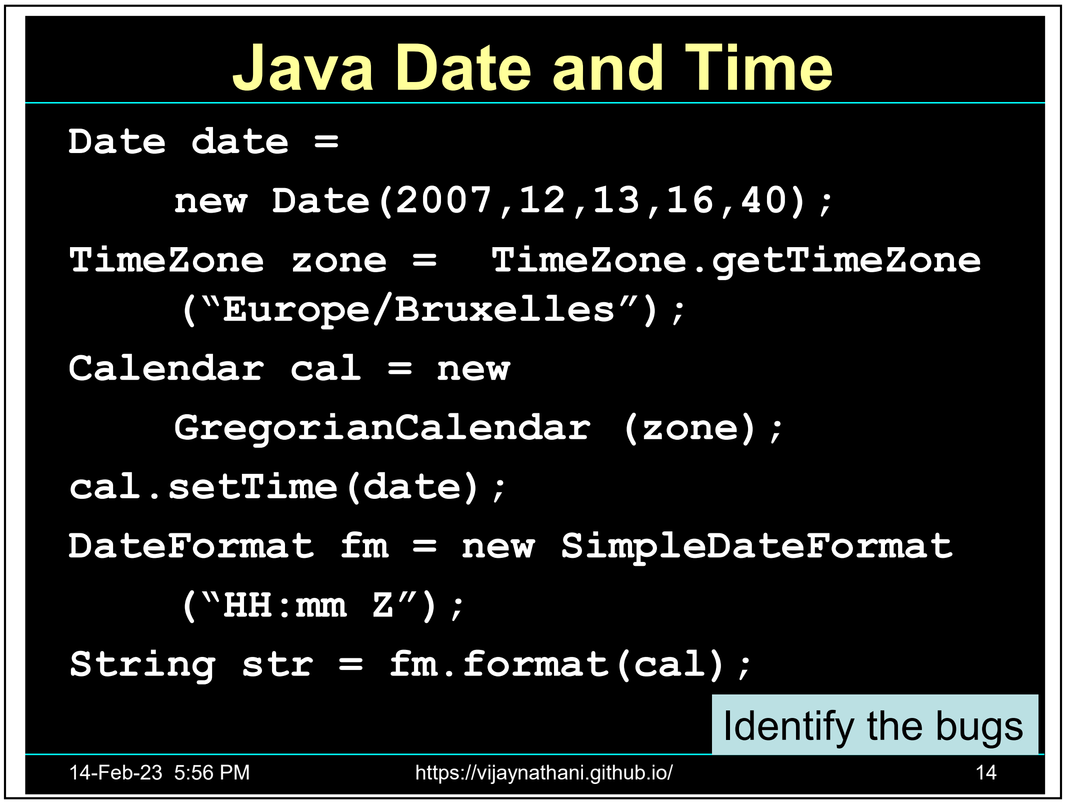
The code on the right

•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

14-Feb-23 5:56 PM https://vijaynathani.github.io/ 14

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

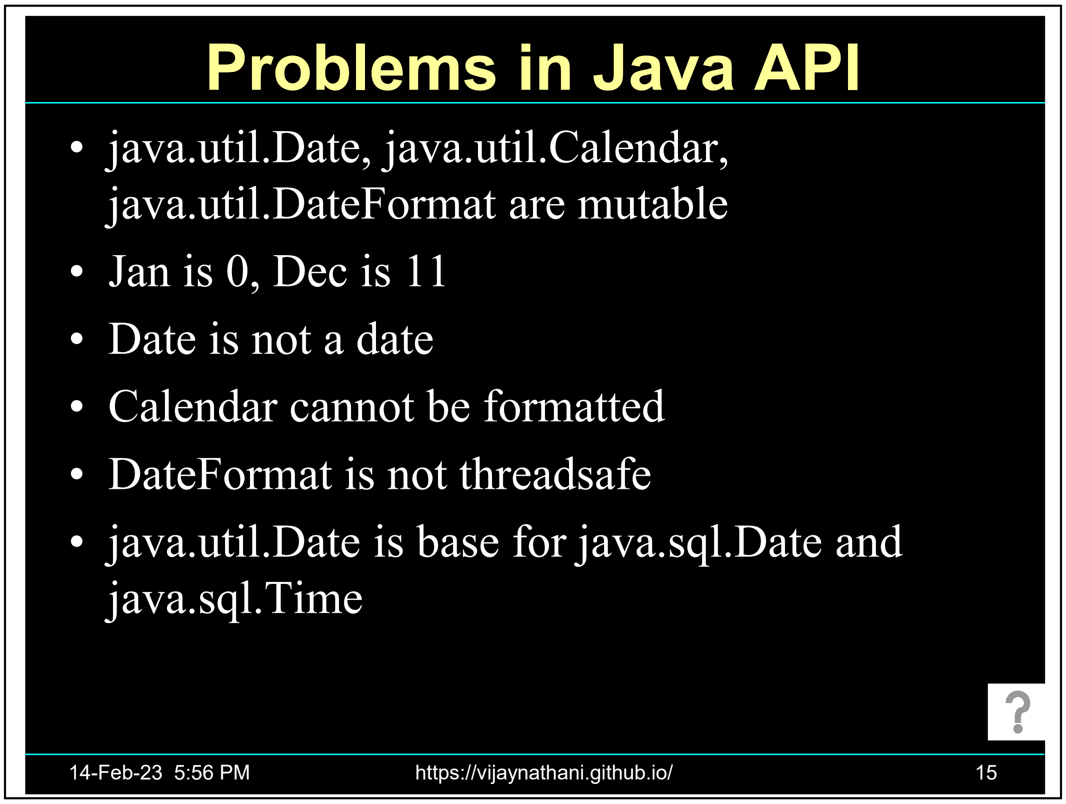
Bug 2: Month should be 11 instead of 12. Since Jan is 0.

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

?

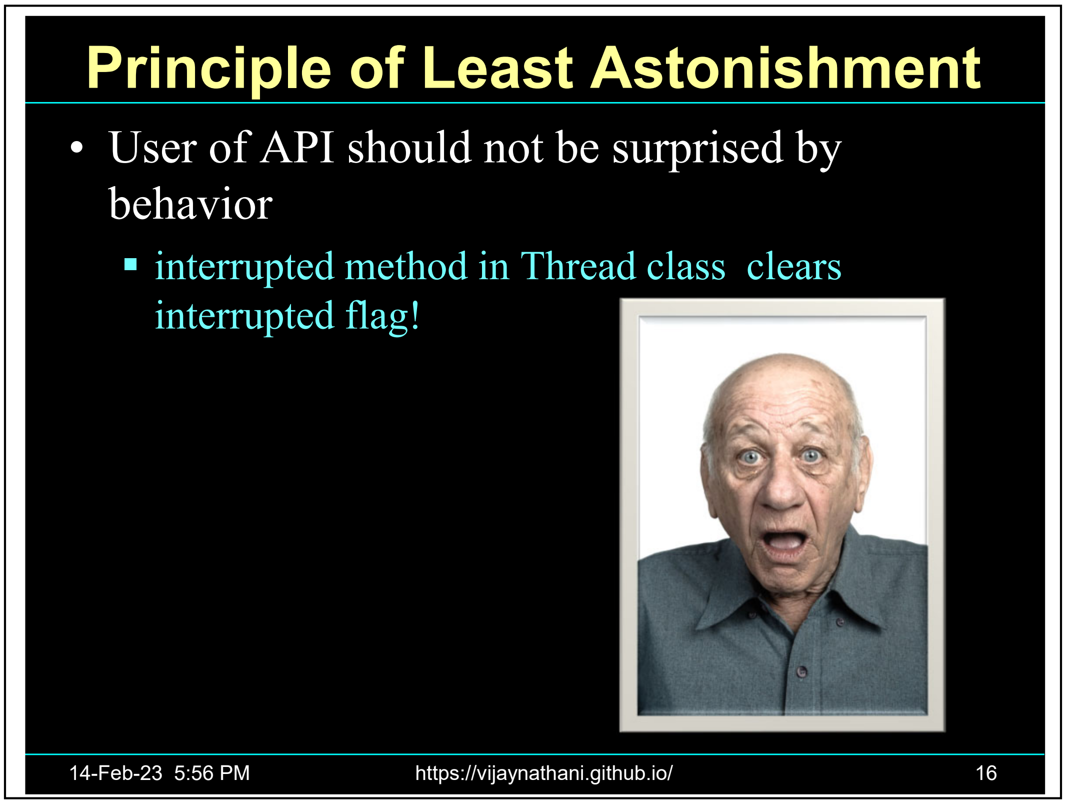
14-Feb-23 5:56 PM https://vijaynathani.github.io/ 15

Q53 – Piano

Q73 – Student, Teacher

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

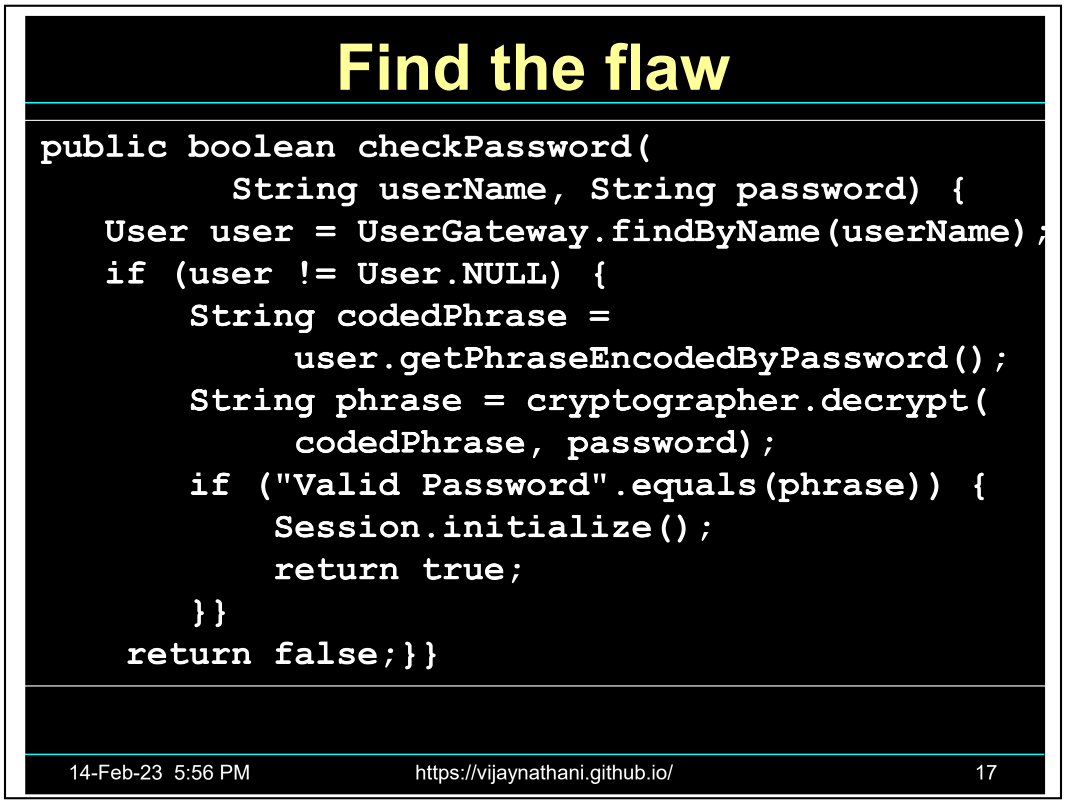
java.util.Date should not base class for java.sql.Date and Time because getYear on java.sql.Time throws an illegal argument exception.



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

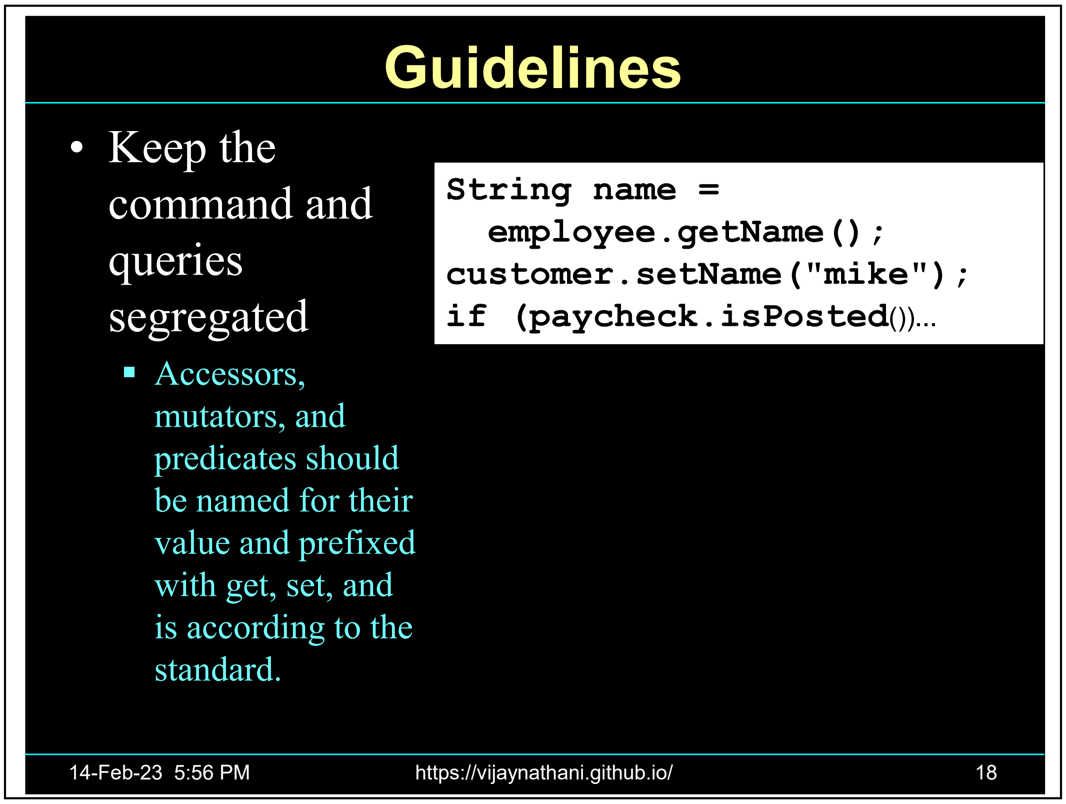
```
public boolean checkPassword(
         String userName, String password) {
   User user = UserGateway.findByName(userName);
   if (user != User.NULL) {
       String codedPhrase =
            user.getPhraseEncodedByPassword();
       String phrase = cryptographer.decrypt(
            codedPhrase,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

14-Feb-23 5:56 PM https://vijaynathani.github.io/ 17

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

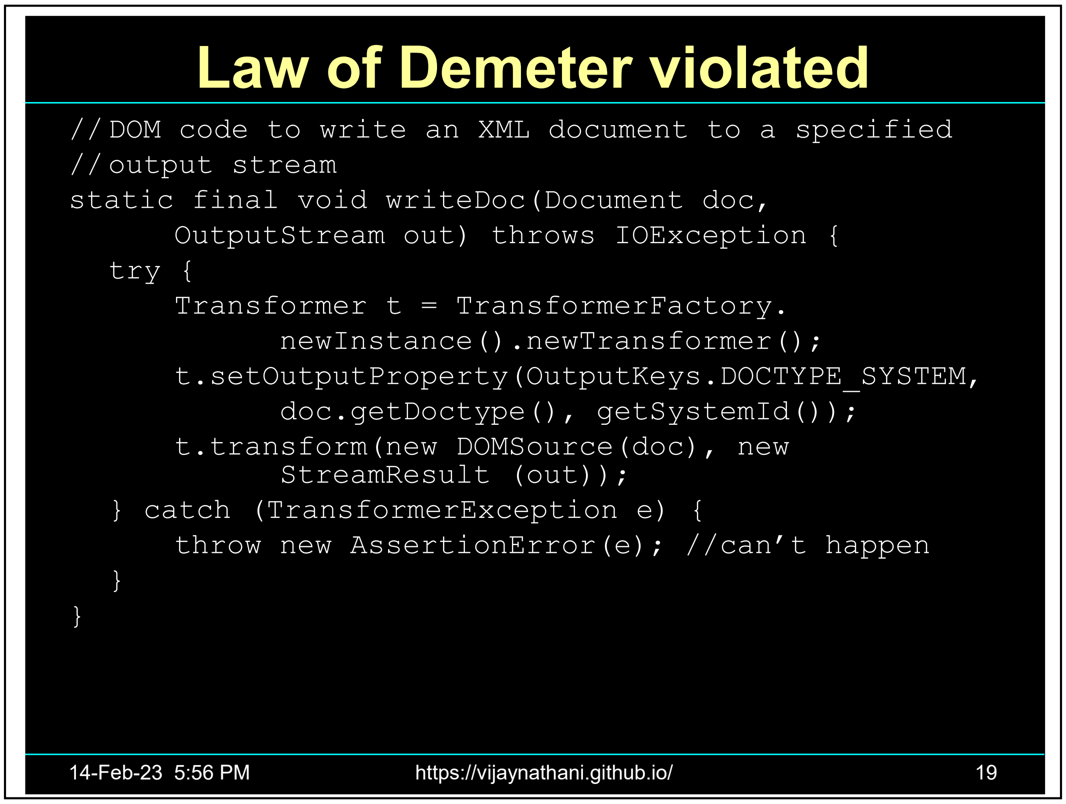
- Keep the command and queries segregated
  - Accessors, mutators, and predicates should be named for their value and prefixed with get, set, and is according to the standard.

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

14-Feb-23 5:56 PM https://vijaynathani.github.io/ 18

Exception: DSL.

Commands : return void.



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
  }
}
```

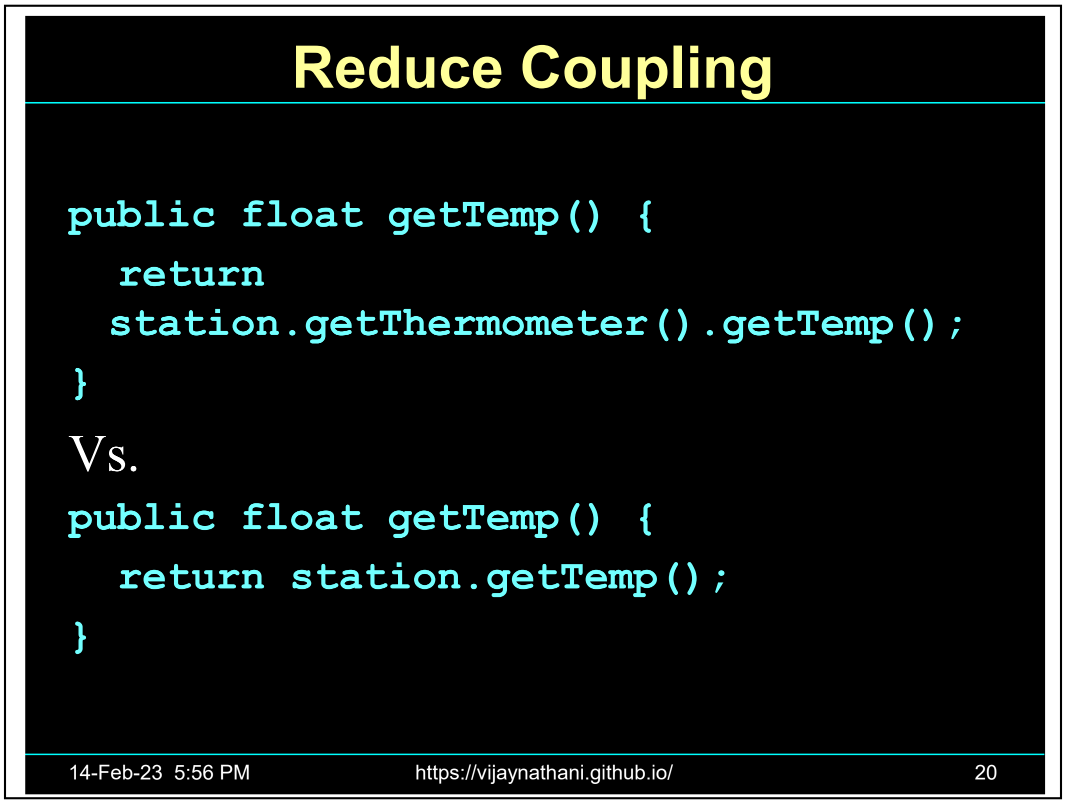
14-Feb-23 5:56 PM https://vijaynathani.github.io/ 19

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



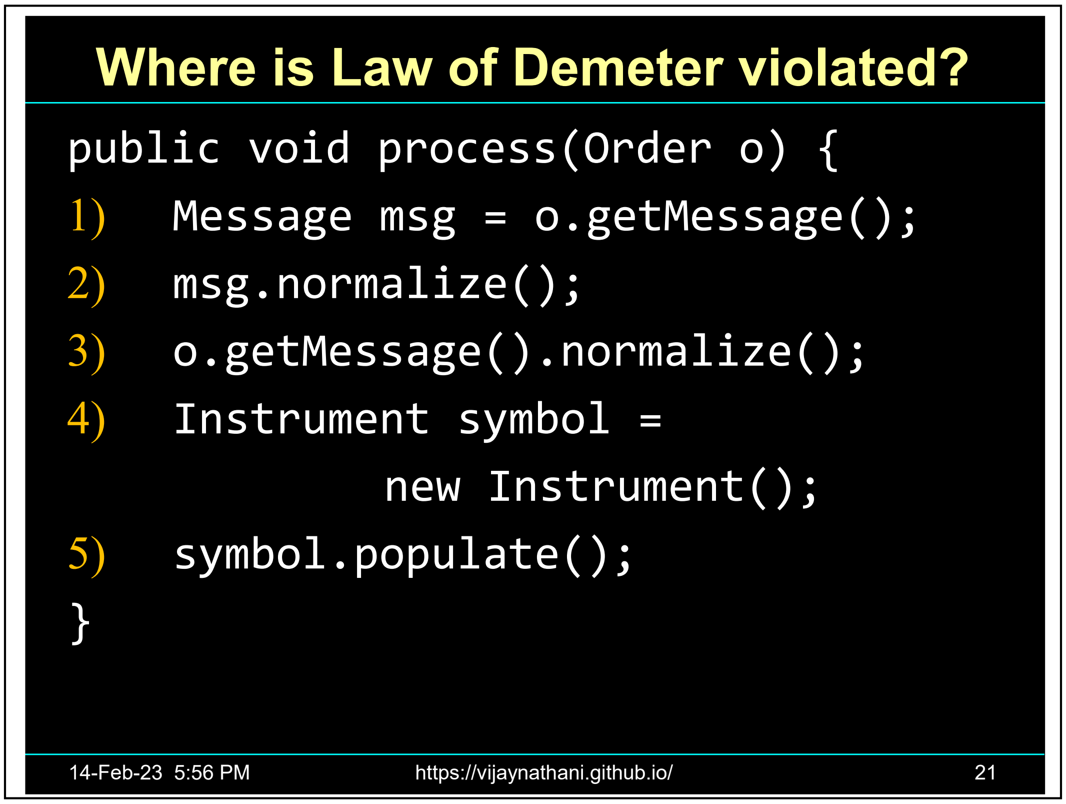
Reduce Coupling

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

Vs.

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

14-Feb-23 5:56 PM https://vijaynathani.github.io/ 20

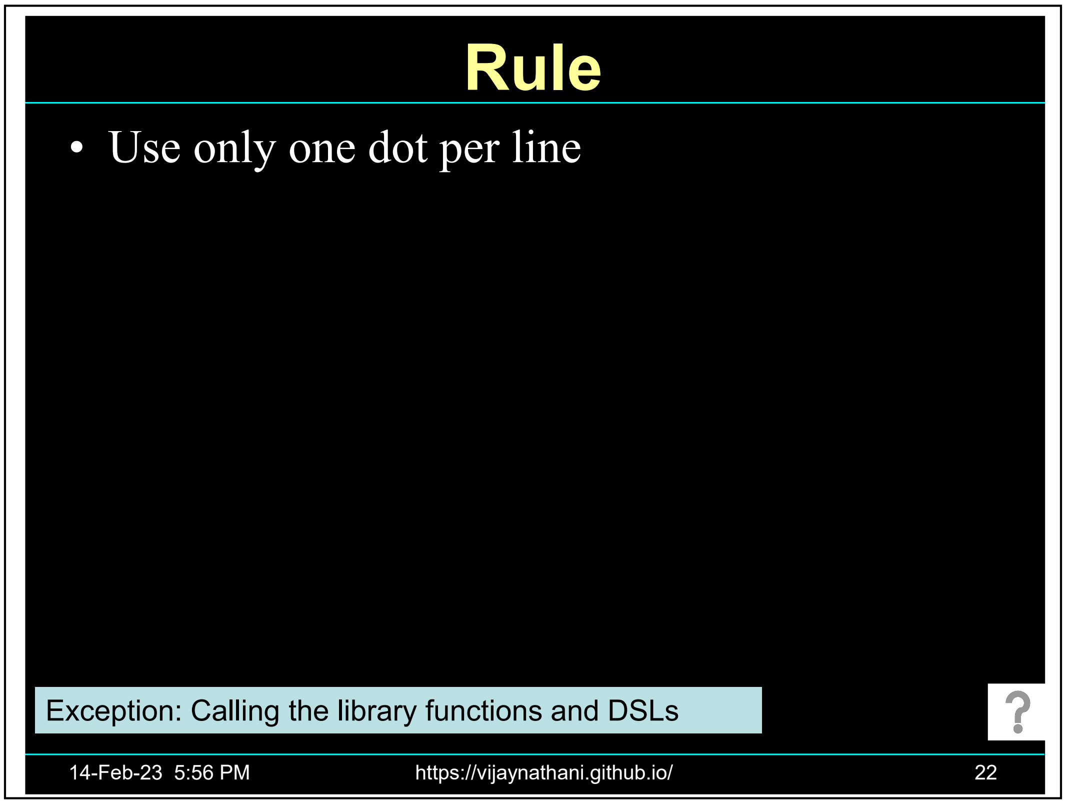


Where is Law of Demeter violated?

```
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();
}
```

14-Feb-23 5:56 PM https://vijaynathani.github.io/ 21

Answer) lines 2 and 3



Rule

- Use only one dot per line

Exception: Calling the library functions and DSLs

14-Feb-23 5:56 PM https://vijaynathani.github.io/ 22

Q26Demeter.