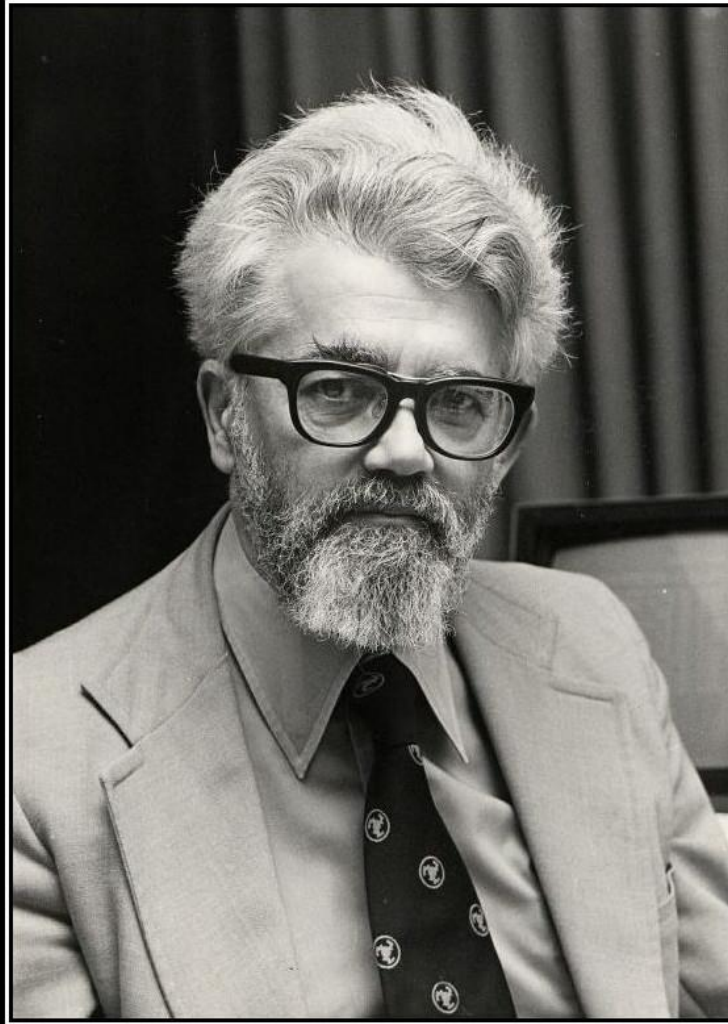


Immutability FTW!

@KevlinHenney

(I told you so...)

@KevlinHenney



PROGRAMMING

YOU'RE DOING IT COMPLETELY WRONG.

Change is the
only constant.

Heraclitus

You cannot step
twice into the
same river.

Heraclitus

Time is like a river,
but frozen not flowing.
Eddies, pools and falls
are fixed in place,
timeless and immutable.

Kevlin Henney

"Remembrance of Things Past"

<http://www.spec-fiction.ca/remembrance-of-things-past/>

When it is not
necessary to
change, it is
necessary not to
change.

Lucius Cary

const

LSP

A type hierarchy is composed of subtypes and supertypes. The intuitive idea of a subtype is one whose objects provide all the behavior of objects of another type (the supertype) plus something extra. What is wanted here is something like the following substitution property: If for each object o_1 of type S there is an object o_2 of type T such that for all programs P defined in terms of T , the behavior of P is unchanged when o_1 is substituted for o_2 , then S is a subtype of T .

Barbara Liskov

"Data Abstraction and Hierarchy"

A type hierarchy is composed of subtypes and supertypes. The intuitive idea of a subtype is one whose objects provide all the behavior of objects of another type (the supertype) plus something extra. What is wanted here is something like the following substitution property: If for each object o_1 of type S there is an object o_2 of type T such that **for all programs P defined in terms of T , the behavior of P is unchanged when o_1 is substituted for o_2** , then S is a subtype of T .

Barbara Liskov

"Data Abstraction and Hierarchy"

A type hierarchy is composed of subtypes and supertypes. The intuitive idea of a subtype is one whose objects provide all the behavior of objects of another type (the supertype) plus something extra. What is wanted here is something like the following substitution property: If for each object o_1 of type S there is an object o_2 of type T such that for all programs P defined in terms of T , the behavior of P is **unchanged** when o_1 is substituted for o_2 , then S is a subtype of T .

Barbara Liskov

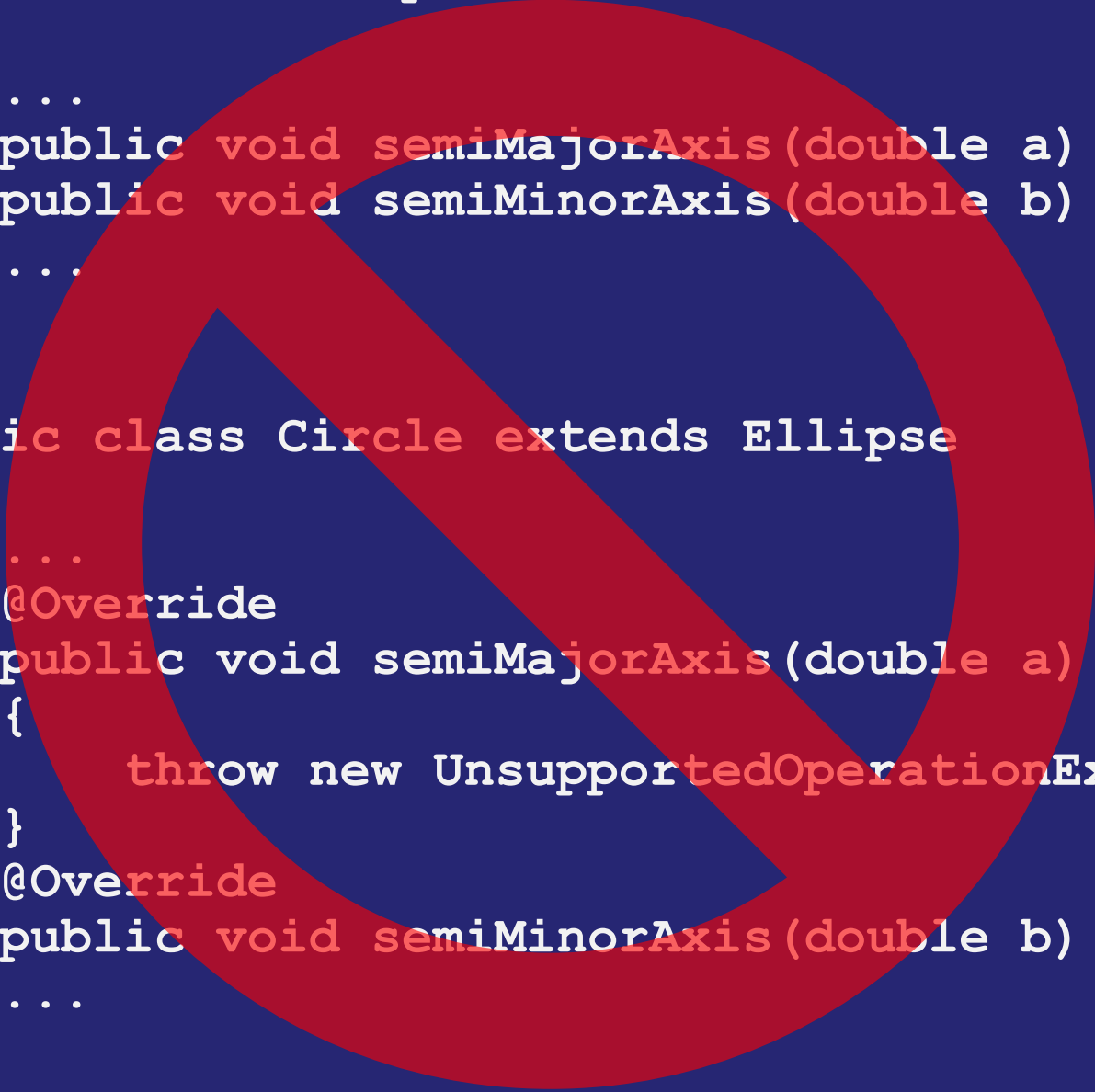
"Data Abstraction and Hierarchy"

```
public class Ellipse
{
    private double semiMajor, semiMinor;
    public Ellipse(double a, double b) ...
    public double semiMajorAxis() ...
    public double semiMinorAxis() ...
    public void semiMajorAxis(double a) ...
    public void semiMinorAxis(double b) ...
    ...
}
```

```
public class Circle extends Ellipse
{
    public Circle(double r) ...
    public double radius() ...
    public void radius(double r) ...
    ...
}
```

```
public class Ellipse
{
    ...
    public void semiMajorAxis(double a) ...
    public void semiMinorAxis(double b) ...
    ...
}

public class Circle extends Ellipse
{
    ...
    @Override
    public void semiMajorAxis(double a)
    {
        throw new UnsupportedOperationException();
    }
    @Override
    public void semiMinorAxis(double b) ...
    ...
}
```



The reason a solution is so hard to come by is because the problem is poorly stated: mathematics tells us that a circle is an ellipse, so I can substitute a circle wherever an ellipse is required, suggesting that a circle is a subtype of an ellipse.

Kevlin Henney

"Vicious Circles", *Overload* 8, June 1995

The reason a solution is so hard to come by is because the problem is poorly stated: mathematics tells us that a circle is an ellipse, so I can substitute a circle wherever an ellipse is required, suggesting that a circle is a subtype of an ellipse.

The troubles start when we introduce any state modifying functions, such as assignment or the ability to change the major and minor axes independently.

Kevlin Henney

"Vicious Circles", *Overload* 8, June 1995

The reason a solution is so hard to come by is because the problem is poorly stated: mathematics tells us that a circle is an ellipse, so I can substitute a circle wherever an ellipse is required, suggesting that a circle is a subtype of an ellipse.

The troubles start when we introduce any state modifying functions, such as assignment or the ability to change the major and minor axes independently.

We are so confident that we understand the mathematical concepts behind circles and ellipses that we have not bothered to ask any more questions of that domain.

Kevlin Henney

"Vicious Circles", *Overload* 8, June 1995

The first observation is that there is no way to change circles and ellipses once you have created them.

Kevlin Henney

"Vicious Circles", *Overload* 8, June 1995

The first observation is that there is no way to change circles and ellipses once you have created them.

This is the correct mathematical model: there are no side effects in maths, conic sections do not undergo state changes, and there are no variables in the programming sense of the word.

Kevlin Henney

"Vicious Circles", *Overload* 8, June 1995

The first observation is that there is no way to change circles and ellipses once you have created them.

This is the correct mathematical model: there are no side effects in maths, conic sections do not undergo state changes, and there are no variables in the programming sense of the word.

Readers who are comfortable and familiar with functional programming and data flow models will recognise the approach.

Kevlin Henney

"Vicious Circles", *Overload* 8, June 1995

The first observation is that there is no way to change circles and ellipses once you have created them.

This is the correct mathematical model: there are no side effects in maths, conic sections do not undergo state changes, and there are no variables in the programming sense of the word.

Readers who are comfortable and familiar with functional programming and data flow models will recognise the approach.

In the case of circles and ellipses, the circle is simply an ellipse with specialised invariants. There is no additional state and none of the members of an ellipse need overriding as they apply equally well to a circle.

Kevlin Henney

"Vicious Circles", *Overload* 8, June 1995

```
public class Ellipse
{
    private double semiMajor, semiMinor;
    public Ellipse(double a, double b) ...
    public double semiMajorAxis() ...
    public double semiMinorAxis() ...
    ...
}
```

```
public class Circle extends Ellipse
{
    public Circle(double r) ...
    public double radius() ...
    ...
}
```

Pure Interface Layer

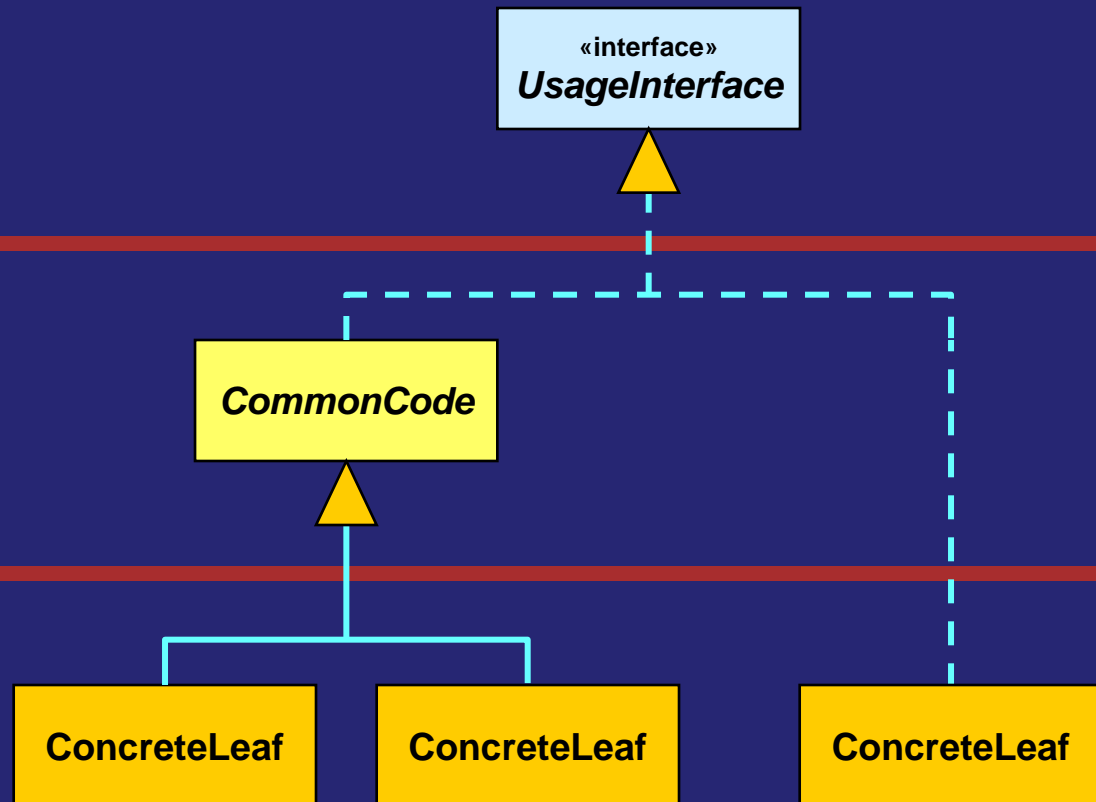
Interfaces may extend interfaces, but there is no implementation defined in this layer.

Common Code Layer

Only abstract classes are defined in this layer, possibly with inheritance, factoring out any common implementation.

Concrete Class Layer

Only concrete classes are defined, and they do not inherit from one another.



```
public interface Ellipse
{
    double semiMajorAxis();
    double semiMinorAxis();
    ...
}
```

```
public interface Circle extends Ellipse
{
    double radius();
    ...
}
```



```
public class ??? implements Ellipse
{
    private double semiMajorAxis, semiMinorAxis;
    ...
}
```

```
public class ??? implements Circle
{
    private double radius;
    ...
}
```

```
public class ??? implements Ellipse
```

```
{
```

*The Naming of Cats is a difficult matter,
It isn't just one of your holiday games;
You may think at first I'm as mad as a hatter
When I tell you, a cat must have THREE DIFFERENT NAMES.*

[...]

*But above and beyond there's still one name left over,
And that is the name that you never will guess;
The name that no human research can discover—
But THE CAT HIMSELF KNOWS, and will never confess.*

[...]

T S Eliot

```
...  
}
```

```
public class Ellipse
{
    private double semiMajor, semiMinor;
    public Ellipse(double a, double b) ...
    public double semiMajorAxis() ...
    public double semiMinorAxis() ...
    ...
}
```

```
public class Circle
{
    private double radius;
    public Circle(double r) ...
    public double radius() ...
    public Ellipse toEllipse() ...
    ...
}
```

```
public class Ellipse
{
    private double semiMajor, semiMinor;
    public Ellipse(double a, double b) ...
    public double semiMajorAxis() ...
    public double semiMinorAxis() ...
    public boolean isCircle() ...
    ...
}
```

C++ REPORT

THE INTERNATIONAL AUTHORITY ON C++ DEVELOPMENT

A 101communications Publication

May 2000, Vol 12/No 5
\$7.95 US/\$10 Can/\$12 Int'l

Building Expression Template Enabled Containers: Extending the Technique to Enhance Technology

Replacing Functors to Simplify the Source Without Losing Efficiency

The Effective Use of Different Design Strategies in C++

A Useful Optimization and Its Consequences

How to Do Case-Insensitive String Comparison

Expression TEMPLATES

New This Month!

FROM MECHANISM TO METHOD
Substitutability: A Useful Way to Structure System Meaning

COMMON KNOWLEDGE
Legitimate Uses of Runtime Behavior

Conversions
Overloading
Derivation
Genericity
Mutability

Phenomenon: An element of what we can observe in the world. Phenomena may be individuals or relations. Individuals are entities, events, or values. Relations are roles, states, or truths.

Individual: An individual is a phenomenon that can be named and is distinct from every other individual: for example, the number 17, George III, or Deep Blue's first move against Kasparov.

Value: A value is an intangible individual that exists outside time and space, and is not subject to change.

beyond



WA
THIS IS A WORK
ALL USERS SHO
CYCLISTS ARE A
No liability will be accep



WILEY SERIES IN
SOFTWARE DESIGN PATTERNS

PATTERN-ORIENTED SOFTWARE ARCHITECTURE

A Pattern Language for
Distributed Computing



Volume 4

Frank Buschmann
Kevlin Henney
Douglas C. Schmidt

Immutable Value

References to value objects are commonly distributed and stored in fields. However, state changes to a value caused by one object can have unexpected and unwanted side-effects for any other object sharing the same value instance. Copying the value can reduce the synchronization overhead, but can also incur object creation overhead.

Therefore:

Define a value object type whose instances are immutable. The internal state of a value object is set at construction and no subsequent modifications are allowed.

```
public class Date implements ...
{
    ...
    public int getYear() ...
    public int getMonth() ...
    public int getDayInMonth() ...
    public void setYear(int newYear) ...
    public void setMonth(int newMonth) ...
    public void setDayInMonth(int newDayInMonth) ...
    ...
}
```

```
public class Date implements ...
{
    ...
    public int getYear() ...
    public int getMonth() ...
    public int getWeekInYear() ...
    public int getDayInYear() ...
    public int getDayInMonth() ...
    public int getDayInWeek() ...
    public void setYear(int newYear) ...
    public void setMonth(int newMonth) ...
    public void setWeekInYear(int newWeek) ...
    public void setDayInYear(int newDayInYear) ...
    public void setDayInMonth(int newDayInMonth) ...
    public void setDayInWeek(int newDayInWeek) ...
    ...
}
```

```
public final class Date implements ...
{
    ...
    public int getYear() ...
    public int getMonth() ...
    public int getWeekInYear() ...
    public int getDayInYear() ...
    public int getDayInMonth() ...
    public int getDayInWeek() ...
    ...
}
```

```
public final class Date implements ...
{
    ...
    public int year() ...
    public int month() ...
    public int weekInYear() ...
    public int dayInYear() ...
    public int dayInMonth() ...
    public int dayInWeek() ...
    ...
}
```


Dictionary **Advanced Search** Results History Bookmarks Options Help Home

get

✓ PRONUNCIATION ✓ SPELLINGS ✓ ETYMOLOGY ✓ QUOTATIONS

Find

get, *n.*¹get, *n.*²get, *n.*³get, *v.*geta, *n. pl.*Getan, *a.* (and *n.*)get-at-able, *a.*get-away, *getaw*gete, *n.*gete, *v.*

gete

getee

geten

getenly, *adv.*

geterne

get-go, *n.*

gethe

gether, *adv.*

gethicall

Gethsemane

get, *v.*

(gɛt)

Pa. tense **got** (*arch.* **gat**). Pa. pple. **got** (**gotten**). Pres. pple. **getting**. Forms: *inf.* 3-4 **geten**, (5 **getyn**), 3-6 **gete**, (4 **geit**, **geyt**, **gite**, *Sc.* **gat(e)**, 4-5 **gyte**, 6 *Sc.* **gait**), 3-7 **gett**, (4-6 **gette**, 4 **gitte**, 5 **gytt**, 9 *dial.* **git**), 3- **get**. *pa. tense* 3-7 **gate**, (3 **gait**, 4 **get**, *pl.* **gaten**, **geton**, -**yn**, **geetun**, **getton**, 5 **geten**), 3-6 **gatt**, (4-6 **gatte**), 3- **gat**, 6- **got**, (6 **got(t)e**). *pa. pple.* α. 3-5 **geten**, (3 **zeten**, **getun**, 4 **getin**, **geteyn**, **giten**, -**in**, **gyten**, -**in**, 4-6 **getyn**, 5 **geton**), 3-5 **getten**, (4-5 **gettyn**, 5 **getton**, 6 **gitten**), 4-6 **gete**, (4 **i-gete**, 5 **y-gete**, **gyte**), 4-6 **gette**, (5 **y-gette**), 5-6 **gett**, (5 **get**). β. 3-4 **gotin**, 3- 6 **goten**, (4 **gotyn**, **gote**, 5 **y-goten**, **goton**, **gothen**), 4-6 *Sc.* **gottin**, -**yn**, 5-7 **gotton**, 6- **gotten**, **got**, (6 **y-got**).

[a. ON. *geta* (*gat*, *gátum*, *getenn*) to get, obtain, to beget, also, to guess (*Sw.* *gitta*, *Da.* *gide* to be able or willing, *Msw.* *gäta*, *Da.* *gjette* to guess) = OE. -*gietan* (only in the compounds *a-*, *be-*, *for-*, *ofer-*, *on-*, *under-gietan*: see **BEGET**, **FORGET**), OFris. (*ur-*, *for-*)*jeta*, OS. (*bi-*, *far-*)*getan* (*Mdu.* *ver-gheten*, *Du.* *ver-geten*), OHG. *ge*<*zced*><*zced*>*an*, *ke*<*zced*><*zced*>*an* (once in pple.

X SORTED BY DATE

Back

Lost for words

Copy

Print

Mark

Find in entry

↑

↓

Dictionary Advanced Search Results History Bookmarks Options Help Home

set

✓ PRONUNCIATION ✓ SPELLINGS ✓ ETYMOLOGY ✓ QUOTATIONS

Find

set, *n.*¹set, *n.*²set, *v.*¹set, *v.*²set, *ppl. a.*set, *conj.*

set-

seta

setace

setaceo-

setaceous, *a.*

setaceous

setal, *a.*

setar

setarious, *a.*set-aside, *n.* and

set-back

setchal, setchel(

set-down

sete, *n.***set**, *v.*¹

(set)

Forms: see below. Pa. tense and pple. **set**.

[Com. Teut.: OE. *settan* = OFris. *setta* (mod.Fris. *sette*), OS. *settian* (MDu., MLG. *setten*, Du. *zetten*), OHG. *sezzen* beside *sazzan* (MHG. *sezzen*, G. *setzen*), ON. *setja* (Sw. *satta*, Da. *sætte*), Goth. *satjan*; causal of **setjan* (*sitjan*) to [sit](#).

Confusion between *set* and *sit* arose as early as the beginning of the 14th c., owing partly to the identity or close similarity of the forms of their past tenses and pa. pples., and partly to the identity of meaning in some uses, as between *to be set* (= seated) and *to sit*; cf. [SIT v.](#) (etym. note and A. 5 a α note). For cases of mere substitution of forms of *sit* for forms of *set*, see A. 1 γ, 2 ζ below. The spelling *sett* is still sometimes found in technical senses; cf. [SET n.](#)¹]

A. Inflectional Forms.

1. a. *inf.* and *pres. stem.* (α) 1 **settan** (Northumb. **setta**), 2-5 (6 *arch.*) **setten**, 3-6 **sette** (2 **setton**, **seotte**, 3 *Orm.* **settenn**, *Lay.* **sætten**, 4 *Kent.* **zetten**, 5 **settyn**, **cettyn**, **satte**, 6 **seatt-**), 4-9

X SORTED BY DATE

Back

Lost for words

Copy

Print

Mark

Find in entry

↑

↓

Dictionary Advanced Search Results History Bookmarks Options Help Home

reset

✓ PRONUNCIATION ✓ SPELLINGS ✓ ETYMOLOGY ✓ QUOTATIONS

Find

reset, *n.*¹

reset, *n.*²

reset, *v.*¹

reset, *v.*²

resetment

resettable, *a.*

resetter

resettle, *v.*

resettlement

reseve

resew, *v.*

resew, *resew*

reseyt

resgat

resh

reshape, *v.*

reshaper

reshare, *v.*

resharpen, *v.*

resheathe, *v.*

reset, *v.*²

(ri:'sɛt)

Also **re-set**.

[RE- 5 a.]

trans. To set again, in various senses of the verb.

1. 1. a. To replace (*esp.* gems) in a (former or new) setting.

1655 FULLER *Ch. Hist.* v. iv. §7 Elizabeth,..finding so fair a flower..fallen out of her Crown, was careful quickly to gather it up again, and get it re-sett therein. **1684** R. WALLER *Nat. Exper. Pref.*, For a time they fall out of their Collets.., and [are] worth nothing till..they are again reset in their proper places. **1830** LYTON *P. Clifford* xix, A stray trinket or two—not of sufficient worth to be re-set. **1883** HALDANE *Worksh. Rec. Ser. II.* 371/2 The hair can be again reset as firmly as it was before [etc.].

b. Surg. To set (a broken limb) again.

X SORTED BY DATE

Back

Lost for words

Copy

Print

Mark

Find in entry

↑

↓

unset

 PRONUNCIATION
 SPELLINGS
 ETYMOLOGY
 QUOTATIONS

Find

unset, v.

unset, *ppl. a.*unsete, *a.*

unsett

unsettling, *ppl. a.*

unsettle, v.

unsettleable, *a.*unsettled, *ppl. a.*

unsettledness

unsettlement

unsettling, *vbl. n.*unsety, *a.*

unseven, v.

unsever, v.

unseverable, *a.*unseverably, *adv.*unsevere, *a.*unsevered, *ppl. a.*

unsew, v.

unsewed, *ppl. a.***un'set**, v.[UN-² 3, 7. Cf. OE. *unsettan* (once), to take down.]**1. trans.** To put out of place or position; to undo the setting of.

1602 MARSTON *Ant. & Mel.* iii. Wks. 1856 l. 37 O, you spoyle my ruffe, unset my haire. **1611** COTGR., *Desplanter*, ..to vnplant, vnset, remoue. **1761** GRAY *Lett.* (1900) ll. 204 The man was sent for: he unset it; it was a paste not worth 40 shillings. **1775** MRS. DELANY in *Life & Corr.* Ser. ii. (1862) ll. 105 There is some hazard in unsettling enamel for fear of chipping the edges. **1836** MARRYAT *Midsh. Easy* xxxii, How could he put the young men to fresh tortures by removing splints and unsettling limbs? **1884** *Law Times* 1 Nov. 8/1 On the morning in question Dawson had unset the gun.

2. intr. To get out of place or position.

1703 THORESBY *Let. to Ray, Spelk*, a wooden splinter tied on, to keep a broken bone from bending or unsettling again.

Copied Value

Value objects are commonly distributed and stored in fields. If value objects are shared between threads, however, state changes caused by one object to a value can have unexpected and unwanted side effects for any other object sharing the same value instance. In a multi-threaded environment shared state must be synchronized between threads, but this introduces costly overhead for frequent access.

Frank Buschmann
Kevin Henney
Douglas C. Schmidt

Therefore:

Define a value object type whose instances are copyable. When a value is used in communication with another thread, ensure that the value is copied.

```
class date
{
public:
    date(int year, int month, int day_in_month);
    date(const date &);
    date & operator=(const date &);
    ...
    int year() const;
    int month() const;
    int day_in_month() const;
    ...
    void year(int);
    void month(int);
    void day_in_month(int);
    ...
};
```

```
class date
{
public:
    date(int year, int month, int day_in_month);
    date(const date &);
    date & operator=(const date &);
    ...
    int year() const;
    int month() const;
    int day_in_month() const;
    ...
    void set(int year, int month, int day_in_month);
    ...
};
```

```
today.set(2014, 4, 11);
```

```
class date
{
public:
    date(int year, int month, int day_in_month);
    date(const date &);
    date & operator=(const date &);
    ...
    int year() const;
    int month() const;
    int day_in_month() const;
    ...
};
```

```
today = date(2014, 4, 11);
```


97




Collective Wisdom
from the Experts

程序员 应该知道的 97件事

Kevlin Henney 编
李军译 吕骏 审校

O'REILLY®

 电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
<http://www.phei.com.cn>

Referential transparency is a very desirable property: it implies that functions consistently yield the same results given the same input, irrespective of where and when they are invoked. That is, function evaluation depends less—ideally, not at all—on the side effects of mutable state.

程序员
应该知道的
97件事

O'REILLY®

Kevlin Henney 编
李军 译 吕骏 审校

电子工业出版社

"Apply Functional Programming Principles"

Edward Garson

"Apply Functional Programming Principles"

**A book is simply the
container of an idea like
a bottle; what is inside
the book is what matters.**

Angela Carter


```
// "FTL" (Functional Template Library)
// container style

template<typename ValueType>
class container
{
public:
    typedef const ValueType value_type;
    typedef ... iterator;
    ...
    bool empty() const;
    std::size_t size() const;
    iterator begin() const;
    iterator end() const;
    ...
    container & operator=(const container &);
    ...
};
```

```
template<typename ValueType>
class set
{
public:
    typedef const ValueType * iterator;
    ...
    set(std::initializer_list<ValueType> values);
    ...
    bool empty() const;
    std::size_t size() const;

    iterator begin() const;
    iterator end() const;

    iterator find(const ValueType &) const;
    std::size_t count(const ValueType &) const;
    iterator lower_bound(const ValueType &) const;
    iterator upper_bound(const ValueType &) const;
    pair<iterator, iterator> equal_range(const ValueType &) const;
    ...
private:
    ValueType * members;
    std::size_t cardinality;
};
```

```
set<int> c = { 2, 9, 9, 7, 9, 2, 4, 5, 8 };
```

```
template<typename ValueType>
class array
{
public:
    typedef const ValueType * iterator;
    ...
    array(std::initializer_list<ValueType> values);
    ...
    bool empty() const;
    std::size_t size() const;

    iterator begin() const;
    iterator end() const;

    const ValueType & operator[](std::size_t) const;
    const ValueType & front() const;
    const ValueType & back() const;
    const ValueType * data() const;
    ...
private:
    ValueType * elements;
    std::size_t length;
};
```

```
array<int> c = { 2, 9, 9, 7, 9, 2, 4, 5, 8 };
```

```
template<typename ValueType>
class vector
{
public:
    typedef const ValueType * iterator;
    ...
    bool empty() const;
    std::size_t size() const;

    iterator begin() const;
    iterator end() const;

    const ValueType & operator[](std::size_t) const;
    const ValueType & front() const;
    const ValueType & back() const;
    const ValueType * data() const;

    vector popped_front() const;
    vector popped_back() const;
    ...
private:
    ValueType * anchor;
    iterator from, until;
};
```

In computing, a persistent data structure is a data structure that always preserves the previous version of itself when it is modified. Such data structures are effectively immutable, as their operations do not (visibly) update the structure in-place, but instead always yield a new updated structure.

(A persistent data structure is not a data structure committed to persistent storage, such as a disk; this is a different and unrelated sense of the word "persistent.")

http://en.wikipedia.org/wiki/Persistent_data_structure

```
template<typename ValueType>
class vector
{
public:
    typedef const ValueType * iterator;
    ...
    bool empty() const;
    std::size_t size() const;

    iterator begin() const;
    iterator end() const;

    const ValueType & operator[](std::size_t) const;
    const ValueType & front() const;
    const ValueType & back() const;
    const ValueType * data() const;

    vector popped_front() const;
    vector popped_back() const;
    void pop_front();
    void pop_back();
    ...
private:
    ValueType * anchor;
    iterator from, until;
};
```

LISP 1.5 Programmer's Manual

**The Computation Center
and Research Laboratory of Electronics
Massachusetts Institute of Technology**

I still have a deep fondness for the Lisp model. It is simple, elegant, and something with which all developers should have an infatuation at least once in their programming life.

Kevlin Henney

"A Fair Share (Part I)", *CUJ C++ Experts Forum*, October 2002

11sp t

```
template<typename ValueType>
class list
{
public:
    class iterator;
    ...
    std::size_t size() const;
    iterator begin() const;
    iterator end() const;

    const ValueType & front() const;
    list popped_front() const;
    list pushed_front() const;
    void pop_front();
    void push_front(const ValueType &);
    ...
private:
    struct link
    {
        link(const ValueType & value, link * next);
        ValueType value;
        link * next;
    };
    link * head;
    std::size_t length;
};
```

*Hamlet: Yea, from the table of
my memory I'll wipe away all
trivial fond records.*

William Shakespeare
The Tragedy of Hamlet
[Act I, Scene 5]

Garbage collection [...] is optional in C++; that is, a garbage collector is not a compulsory part of an implementation.

Bjarne Stroustrup

<http://stroustrup.com/C++11FAQ.html>

```
assert(  
    std::get_pointer_safety() ==  
    std::pointer_safety::strict);
```

```
public interface RecentlyUsedList
{
    static final RecentlyUsedList nil = new Null();

    boolean isEmpty();
    int size();
    String get(int index);
    RecentlyUsedList add(String newItem);
    RecentlyUsedList remove(String toRemove);
}
```

```
RecentlyUsedList list =
    nil.add("Alice").add("Bob").add("Alice");

assert list.size() == 2;
assert list.get(0).equals("Alice");
assert list.get(1).equals("Bob");
```

Ophelia: 'Tis in my memory
locked, and you yourself shall
keep the key of it.

William Shakespeare
The Tragedy of Hamlet
[Act I, Scene 3]

A use-counted class is more complicated than a non-use-counted equivalent, and all of this horsing around with use counts takes a significant amount of processing time.

Robert Murray
C++ Strategies and Tactics


```
template<typename ValueType>
class vector
{
public:
    typedef const ValueType * iterator;
    ...
    bool empty() const;
    std::size_t size() const;

    iterator begin() const;
    iterator end() const;

    const ValueType & operator[](std::size_t) const;
    const ValueType & front() const;
    const ValueType & back() const;
    const ValueType * data() const;

    vector popped_front() const;
    vector popped_back() const;
    void pop_front();
    void pop_back();
    ...
private:
    std::shared_ptr<ValueType> anchor;
    iterator from, until;
};
```

Uses *std::default_delete<ValueType[]>*, but cannot be initialised from *std::make_shared*.

```
template<typename CharType>  
class string;
```

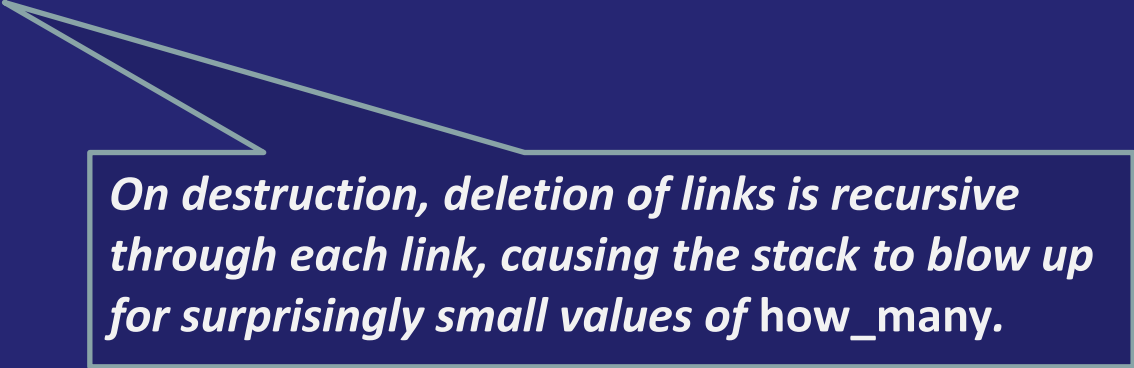
```
template<typename CharType>  
class string<const CharType>;
```

```
template<typename CharType>  
class string;
```

```
template<typename ValueType>
class list
{
public:
    class iterator;
    ...
    std::size_t size() const;
    iterator begin() const;
    iterator end() const;

    const ValueType & front() const;
    list popped_front() const;
    list pushed_front() const;
    void pop_front();
    void push_front(const ValueType &);
    ...
private:
    struct link
    {
        link(const ValueType & value, std::shared_ptr<link> next);
        ValueType value;
        std::shared_ptr<link> next;
    };
    std::shared_ptr<link> head;
    std::size_t length;
};
```

```
{  
    list<Anything> chain;  
    std::fill_n(  
        std::front_inserter(chain),  
        how_many,  
        something);  
}
```



On destruction, deletion of links is recursive through each link, causing the stack to blow up for surprisingly small values of how_many.

Algorithms +
Data Structures =
Programs

Niklaus Wirth

**HOLY GOD
WILL BRING
JUDGMENT
DAY ON
MAY 21, 2011**
**CRY MIGHTILY UNTO
GOD FOR MERCY SEE
PSALMS - 51:
JONAH - 3:**



paraskevidekatriaphobia, *noun*

- The superstitious fear of Friday 13th.
- Contrary to popular myth, this superstition is relatively recent (19th century) and did not originate during or before the medieval times.
- Paraskevidekatriaphobia (or friggatriskaidekaphobia) also reflects a particularly egocentric attributional bias: the universe is prepared to rearrange causality and probability around the believer based on an arbitrary and changeable calendar system, in a way that is sensitive to geography, culture and time zone.

```
struct tm next_friday_13th(const struct tm * after)
{
    struct tm next = *after;
    enum { daily_secs = 24 * 60 * 60 };
    time_t seconds =
        mktime(&next) +
        (next.tm_mday == 13 ? daily_secs : 0);
    do
    {
        seconds += daily_secs;
        next = *localtime(&seconds);
    }
    while(next.tm_mday != 13 || next.tm_wday != 5);
    return next;
}
```

```
std::find_if(
    ++begin, day_iterator(),
    [](const std::tm & day)
    {
        return day.tm_mday == 13 && day.tm_wday == 5;
    });
```

```
class day_iterator : public std::iterator<...>
{
public:
    day_iterator() ...
    explicit day_iterator(const std::tm & start) ...
    const std::tm & operator*() const
    {
        return day;
    }
    const std::tm * operator->() const
    {
        return &day;
    }
    day_iterator & operator++()
    {
        std::time_t seconds = std::mktime(&day) + 24 * 60 * 60;
        day = *std::localtime(&seconds);
        return *this;
    }
    day_iterator operator++(int) ...
    ...
};
```

```
var friday13ths =  
  from day in Days.After(start)  
  where day.Day == 13  
  where day.DayOfWeek == DayOfWeek.Friday  
  select day;  
  
foreach(var irrationalBelief in friday13ths)  
{  
  ...  
}
```



```
public class Days : IEnumerable<DateTime>
{
    public static Days After(DateTime startDay)
    {
        return new Days(startDay.AddDays(1));
    }
    public IEnumerator<DateTime> GetEnumerator()
    {
        for(var next = startDay;; next = next.AddDays(1))
            yield return next;
    }
    ...
    private DateTime startDay;
}
```

Iterator

Clients often want to traverse elements that are encapsulated within an aggregate, such as the elements maintained by a collection. Clients may not wish, however, to depend on the aggregate's internal structure to access components of interest.

Therefore:

Objectify the strategy to access and traverse the components maintained by the aggregate into a separate iterator component. Let this iterator be the only means for clients to access the component.

Enumeration Method

Some types of aggregate [...] have representations that do not conveniently support Iterator-based traversal.

Similarly, using an Iterator approach to access the elements of an aggregate that is shared between threads can incur unnecessary overhead from repeated locking.

Therefore:

Bring the iteration inside the aggregate and encapsulate it in a single enumeration method that is responsible for complete traversal. Pass the task of the loop—the action to be executed on each element of the aggregate—as an argument to the enumeration method, and apply it to each element in turn.

Lifecycle Callback

The lifecycle of some objects is simple: their clients create them before they are used, they stay alive as long as they are used, and they are disposed of by their clients when no longer used. However, some objects have a much more complex lifecycle, driven by the needs and events of their component environment.

Therefore:

Define key lifecycle events as callbacks in an interface that is supported by framework objects. The framework uses the callbacks to control the objects' lifecycle explicitly.

```
public interface IObservable<out T>
{
    IDisposable Subscribe(IObserver<T> observer);
}
```

```
public interface IObserver<in T>
{
    void OnCompleted();
    void OnError(Exception error);
    void OnNext(T value);
}
```

```
        IDisposable subscription =
            source.Subscribe(
                value => handle element,
                error => handle exception,
                () => handle completion);
```


Observer

Consumer objects sometimes depend on the state of, or data maintained by, another provider object. If the state of the provider object changes without notice, however, the state of the dependent consumer objects can become inconsistent.

Therefore:

Define a change-propagation mechanism in which the provider—known as the ‘subject’—notifies registered consumers—known as the ‘observers’—whenever its state changes, so that the notified observers can perform whatever actions they deem necessary.

```
void run(const function tests[], const char * label, listener & sink);  
  
// precondition: tests is a non-null null-terminated array  
// execution:  
// sink.start_testing(label) ->  
// (for each test in tests:  
// (sink.before_test(...) ->  
// (test executed ->  
// (one of:  
// (sink.test_passed(...) |  
// sink.assertion_failed(...) |  
// sink.exception_thrown(...)))  
// sink.after_test(...))) ->  
// sink.finish_testing(label)
```



```

void run(const function tests[], const char * label, listener & sink);

// precondition: tests is a non-null null-terminated array
// execution:
//   sink.start_testing(label) ->
//   (for each test in tests:
//     (sink.before_test(...) ->
//       (test executed ->
//         (one of:
//           (sink.test_passed(...) |
//            sink.assertion_failed(...) |
//            sink.exception_thrown(...)))
//       sink.after_test(...))) ->
//   sink.finish_testing(label)

```

```
sink.start_testing(label) ->
(for each test in tests:
  (sink.before_test(...) ->
    (test executed ->
      (one of:
        (sink.test_passed(...) |
          sink.assertion_failed(...) |
          sink.exception_thrown(...)))
      sink.after_test(...))) ->
sink.finish_testing(label)
```

```
class listener
{
public:
    ... start_testing(...) = 0;
    ... before_test(...) = 0;

    ... test_passed(...) = 0;
    ... assertion_failed(...) = 0;
    ... exception_thrown(...) = 0;
    ... after_test(...) = 0;
    ... finish_testing(...) = 0;

};
```

C.A.R. Hoare
**Communicating
Sequential
Processes**

C.A.R. HOARE SERIES EDITOR

Concurrency

Concurrency

Threads

Concurrency

Threads

Locks

Some people, when confronted with a problem, think, "I know, I'll use threads," and then two they hav erpoblesms.

Ned Batchelder

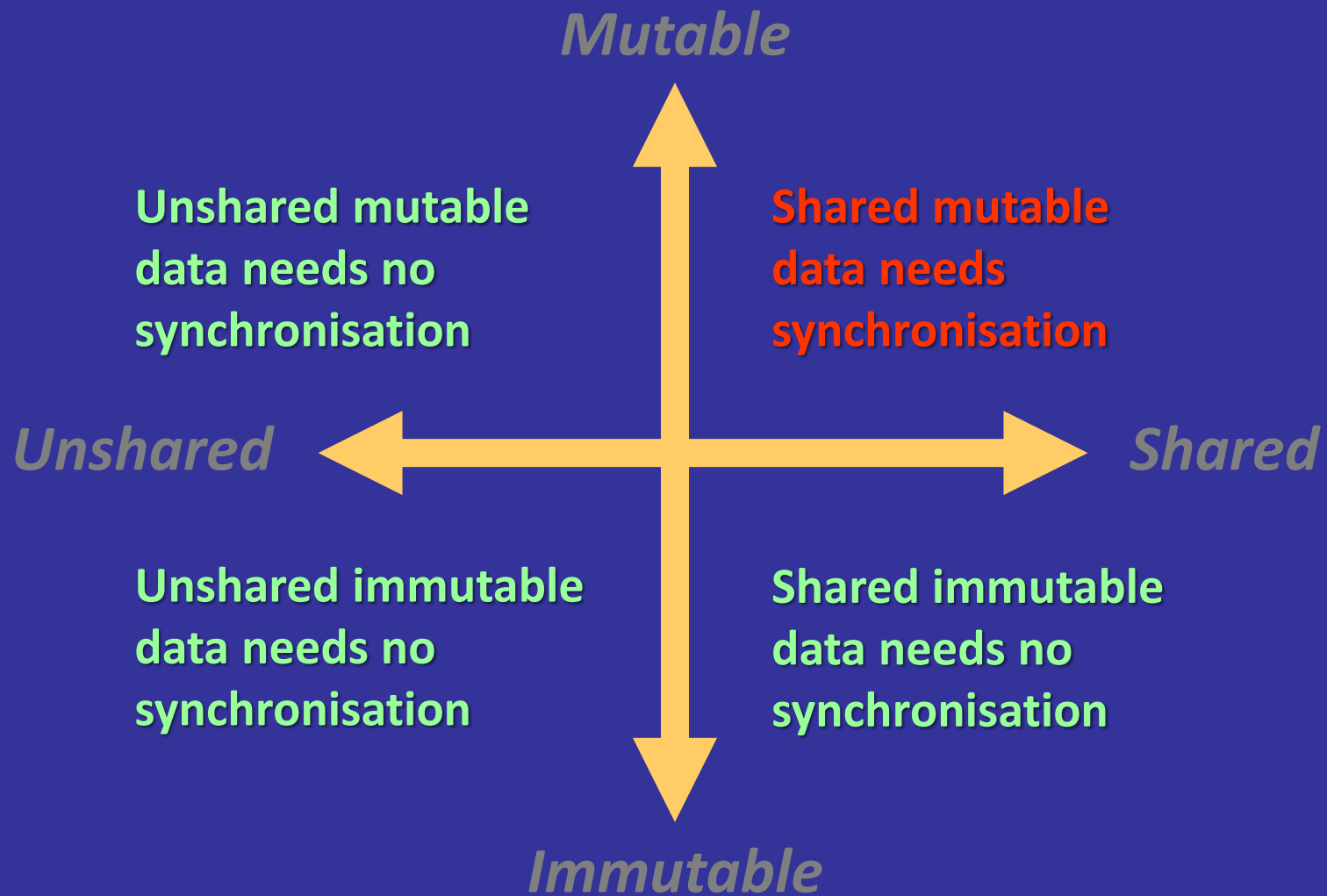
<https://twitter.com/#!/nedbat/status/194873829825327104>

Shared memory is like a canvas where threads collaborate in painting images, except that they stand on the opposite sides of the canvas and use guns rather than brushes. The only way they can avoid killing each other is if they shout "duck!" before opening fire.

Bartosz Milewski

"Functional Data Structures and Concurrency in C++"

<http://bartoszmilewski.com/2013/12/10/functional-data-structures-and-concurrency-in-c/>



Instead of using threads and shared memory as our programming model, we can use processes and message passing. Process here just means a protected independent state with executing code, not necessarily an operating system process.

程序员
应该知道的
97件事

O'REILLY®



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY

Kevlin Henney 编
李军 译 吕骏 审校

"Message Passing Leads to Better Scalability in Parallel Systems"

Russel Winder

OOP to me means only messaging,
local retention and protection and
hiding of state-process, and
extreme late-binding of all things.
It can be done in Smalltalk and in
LISP. There are possibly other
systems in which this is possible,
but I'm not aware of them.

Alan Kay

Languages such as Erlang (and occam before it) have shown that processes are a very successful mechanism for programming concurrent and parallel systems. Such systems do not have all the synchronization stresses that shared-memory, multithreaded systems have.

程序员
应该知道的
97件事

Kevlin Henney 编
李军 译 吕骏 审校

O'REILLY®

电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY

"Message Passing Leads to Better Scalability in Parallel Systems"

Russel Winder

*Computer Systems
Series*

ABCL
*An Object-Oriented Concurrent
System*

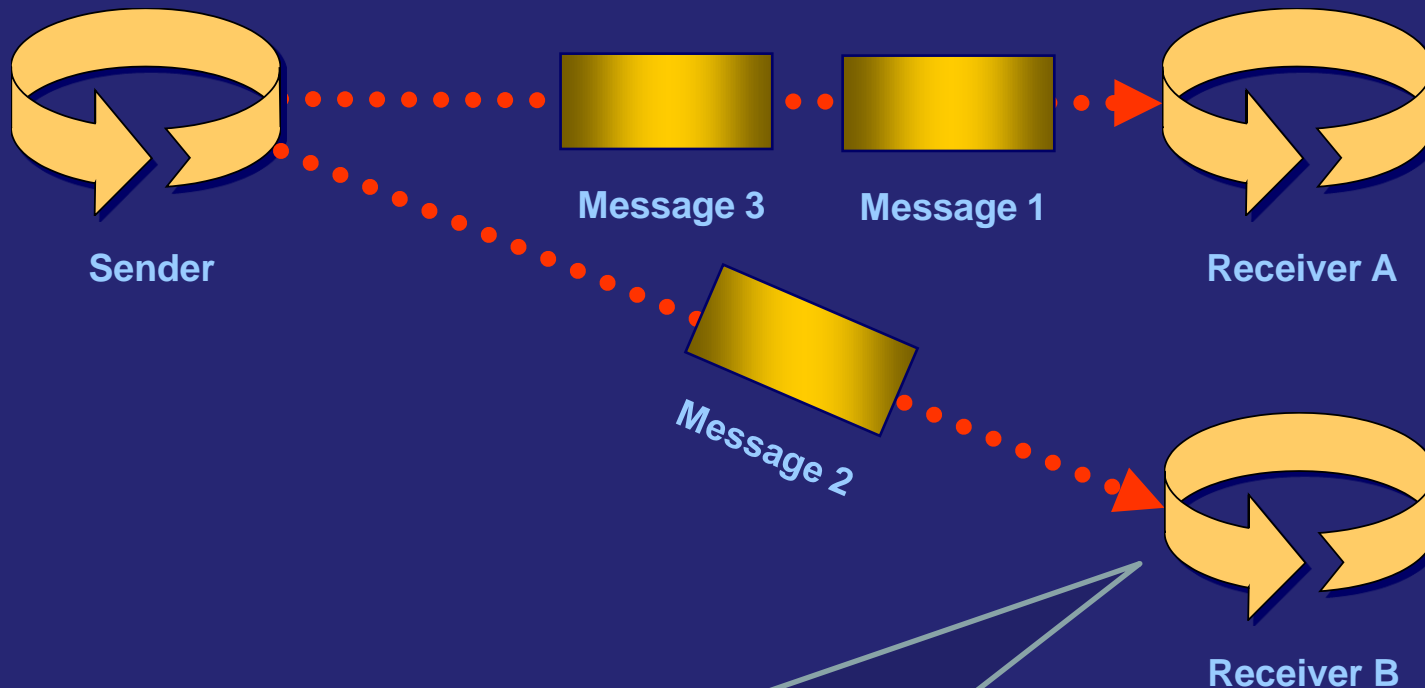
edited by Akinori Yonezawa

The MIT Press

Multithreading is just one damn thing after, before, or simultaneous with another.

Andrei Alexandrescu

Actor-based concurrency is
just one damn message after
another.



In response to a message that it receives, an actor can make local decisions, create more actors, send more messages, and determine how to respond to the next message received.

http://en.wikipedia.org/wiki/Actor_model

Concatenative programming is so called because it uses **function *composition*** instead of **function *application***—a non-concatenative language is thus called *applicative*.

Jon Purdy

<http://evincarofautumn.blogspot.in/2012/02/why-concatenative-programming-matters.html>

$$f(g(h(x)))$$

$$(f \circ g \circ h)(x)$$

x h g f

$$f \circ g \circ h$$

h | *g* | *f*

This is the basic reason Unix pipes are so powerful: they form a rudimentary string-based concatenative programming language.

Jon Purdy

<http://evincarofautumn.blogspot.in/2012/02/why-concatenative-programming-matters.html>

Summary--what's most important.

To put my strongest concerns in a nutshell:

1. We should have some ways of coupling programs like garden hose--screw in another segment when it becomes when it becomes necessary to massage data in another way. This is the way of IO also.
2. Our loader should be able to do link-loading and controlled establishment.
3. Our library filing scheme should allow for rather general indexing, responsibility, generations, data path switching.
4. It should be possible to get private system components (all routines are system components) for bugging around with.

M. D. McIlroy
Oct. 11, 1964

Pipes and Filters

Some applications process streams of data: input data streams are transformed stepwise into output data streams. However, using common and familiar request/response semantics for structuring such types of application is typically impractical. Instead we must specify an appropriate data flow model for them.

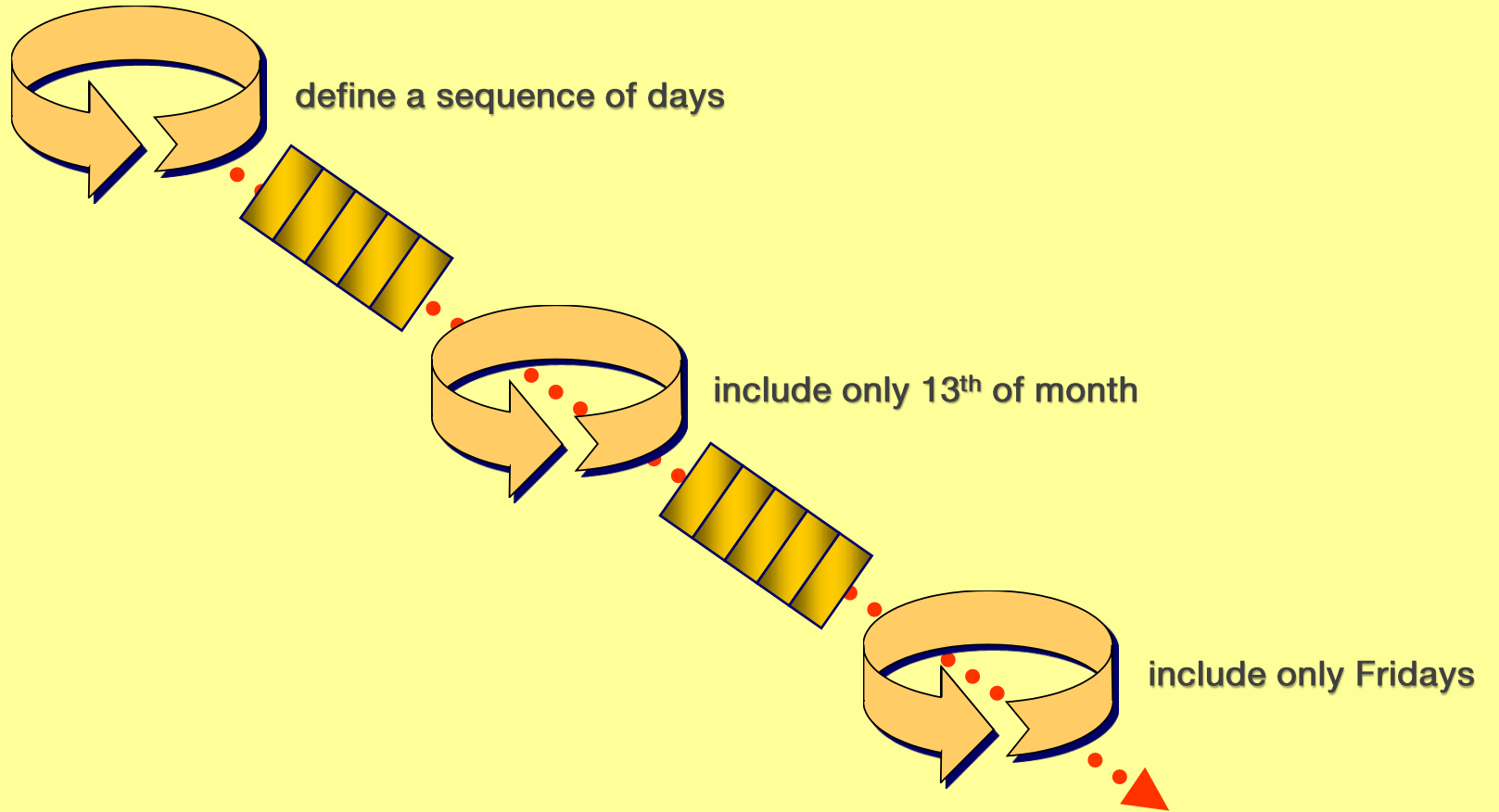
Therefore:

Divide the application's task into several self-contained data processing steps and connect these steps to a data processing pipeline via intermediate data buffers.

Frank Buschmann

Kevin Henney

Douglas C. Schmidt



```
function GetNextFriday13th($from) {  
    [DateTime[]] $friday13ths =  
        (1..500) |  
        %{ $from.AddDays($_) } |  
        ?{ $_.Day -eq 13 } |  
        ?{ $_.DayOfWeek -eq [DayOfWeek]::Friday }  
    return $friday13ths[0]  
}
```

```
[DateTime[] []] $inputsWithExpectations =
    ("2001-07-13", "2002-09-13"),
    ("2007-04-01", "2007-04-13"),
    ("2007-04-12", "2007-04-13"),
    ("2007-04-13", "2007-07-13"),
    ("2011-01-01", "2011-05-13"),
    ("2011-05-13", "2012-01-13"),
    ("2012-01-01", "2012-01-13"),
    ("2012-01-13", "2012-04-13"),
    ("2012-04-13", "2012-07-13"),
    ("2014-04-11", "2014-06-13")

$inputsWithExpectations | ?{
    [String] $actual = GetNextFriday13th($_[0])
    [String] $expected = $_[1]
    $actual -ne $expected
}
```

Everything
flows, nothing
stands still.

Heraclitus





Go with
the Flow.

Queens of the Stone Age