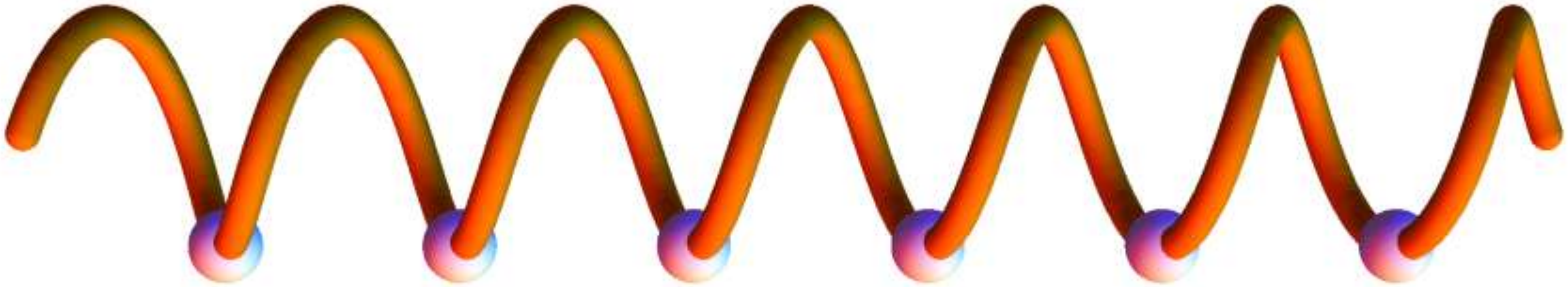


How functional programming made me a better OO developer

Jessica Kerr

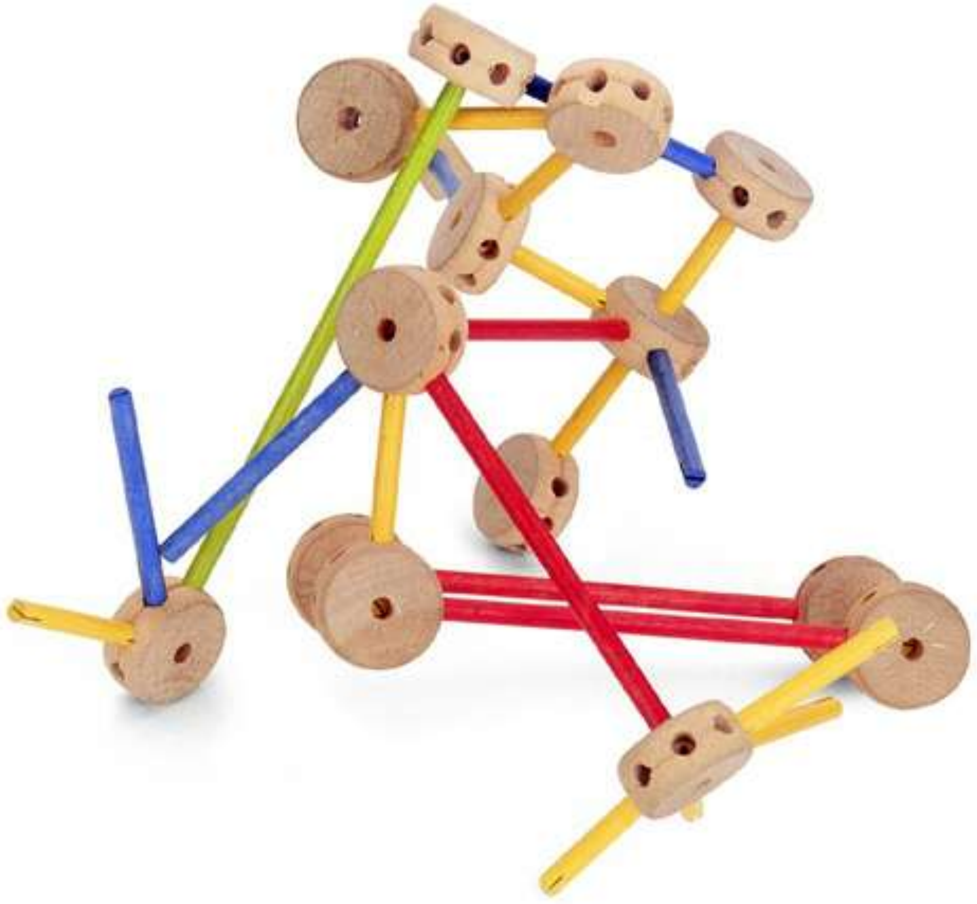
@jessitron

developer’s creed

I am more than an Object-Oriented Developer.

I am a solver of problems, a creator of solutions.

What do we love about OO?

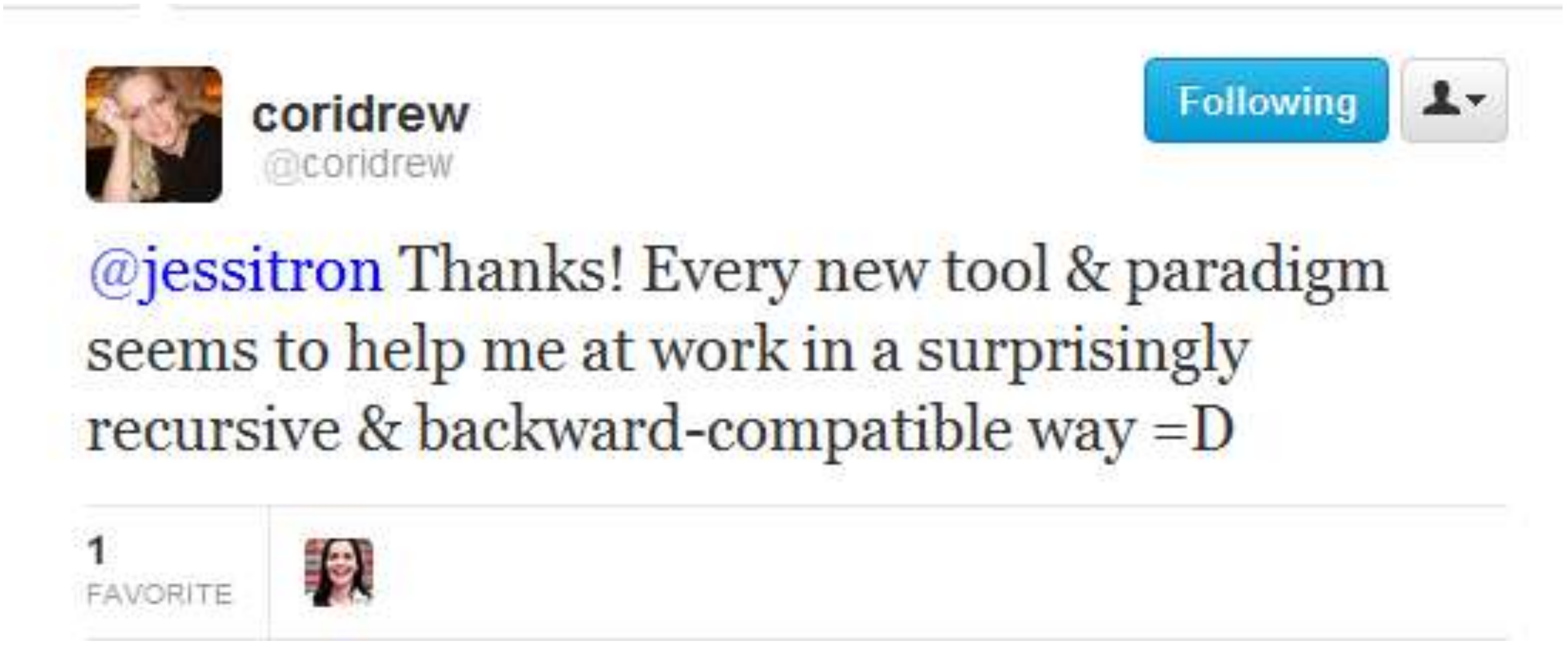


Functional programming will solve all our problems

No.

Programming paradigms

- Imperative
- Procedural
- Object-Oriented
- Functional
- Aspect-oriented
- Logic



coridrew
@coridrew

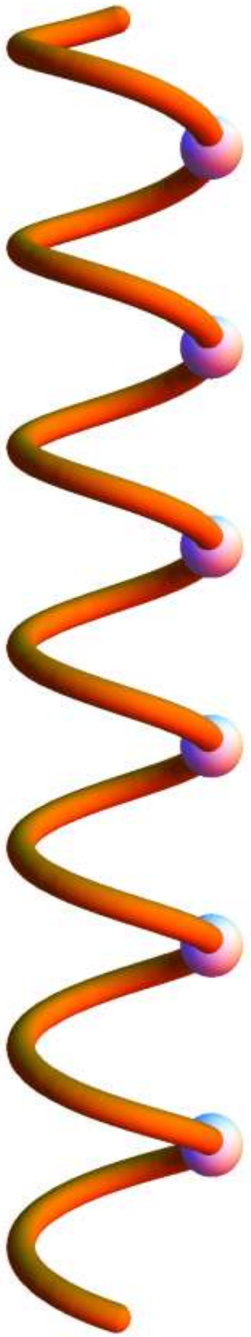
Following

@jessitron Thanks! Every new tool & paradigm seems to help me at work in a surprisingly recursive & backward-compatible way =D

1
FAVORITE

Goals for today

1) Look at functional principles
2) Learn how functional programmers solve problems
3) Solve more problems



Immutability

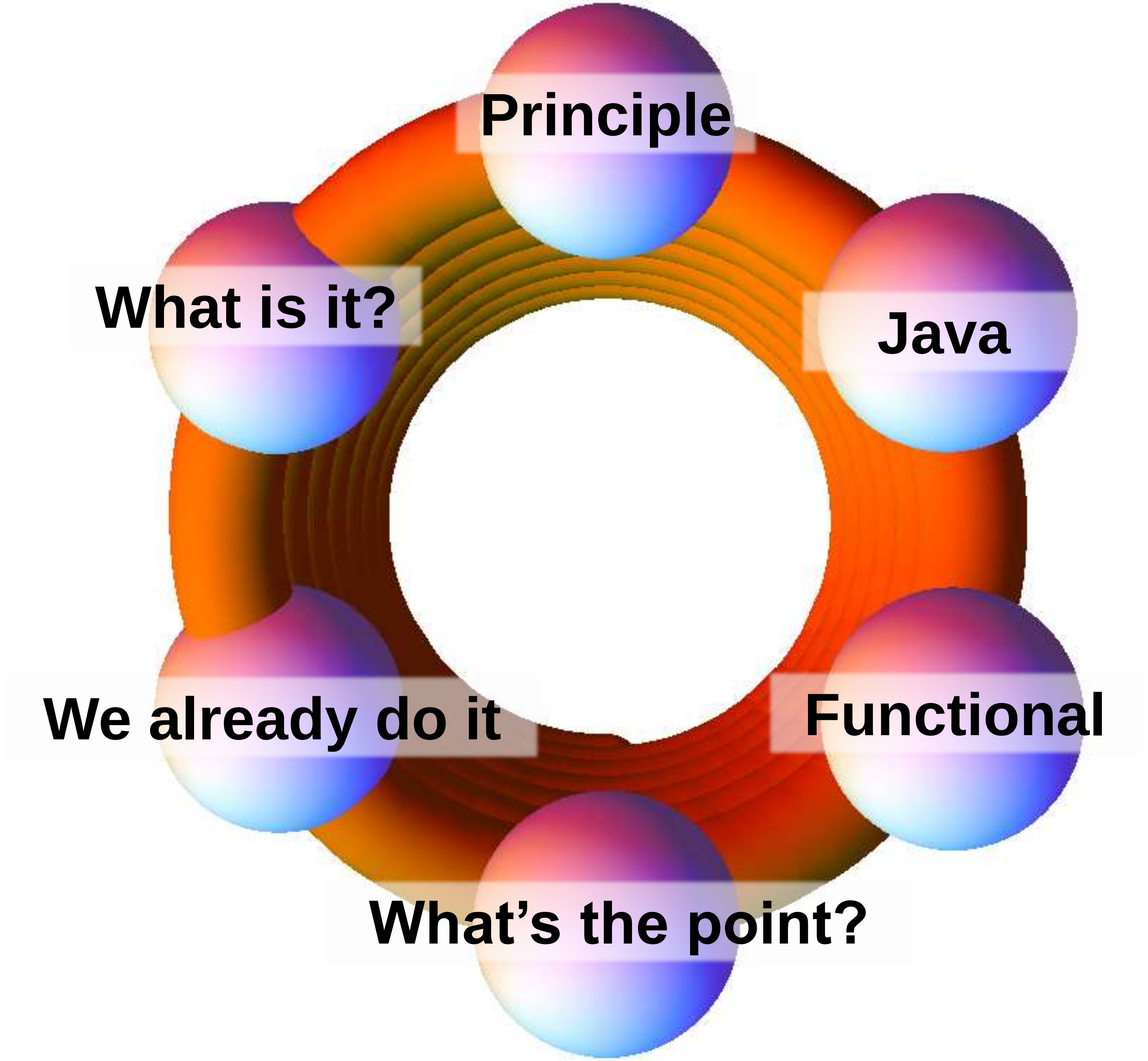
Verbs Are People Too

Declarative Style

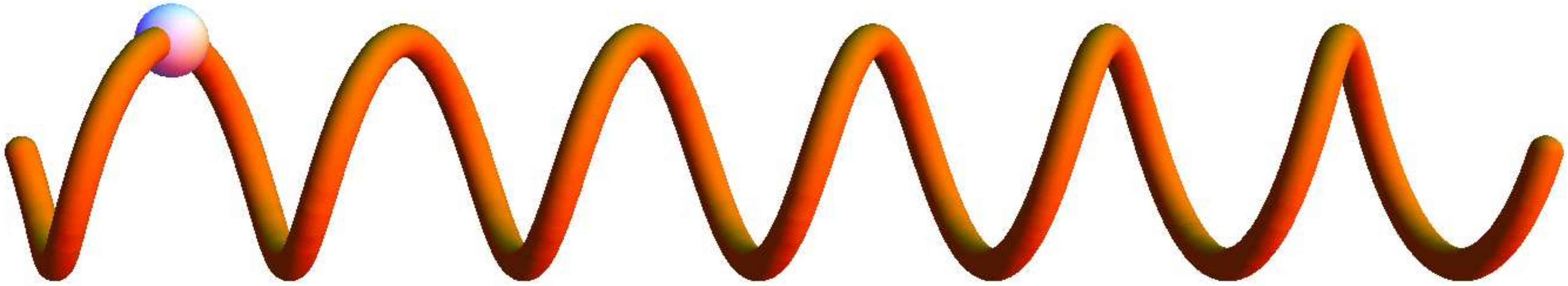
Null Is Your Enemy

Strong Typing

Lazy Evaluation


Principle
Java
Functional
What's the point?
We already do it
What is it?

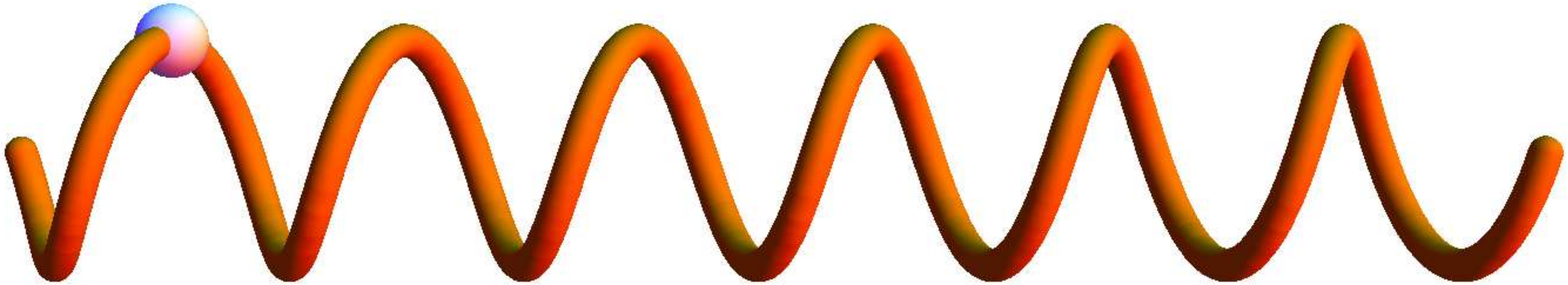
Immutability



What is it?

The value of an identifier never changes.

Objects never change state.

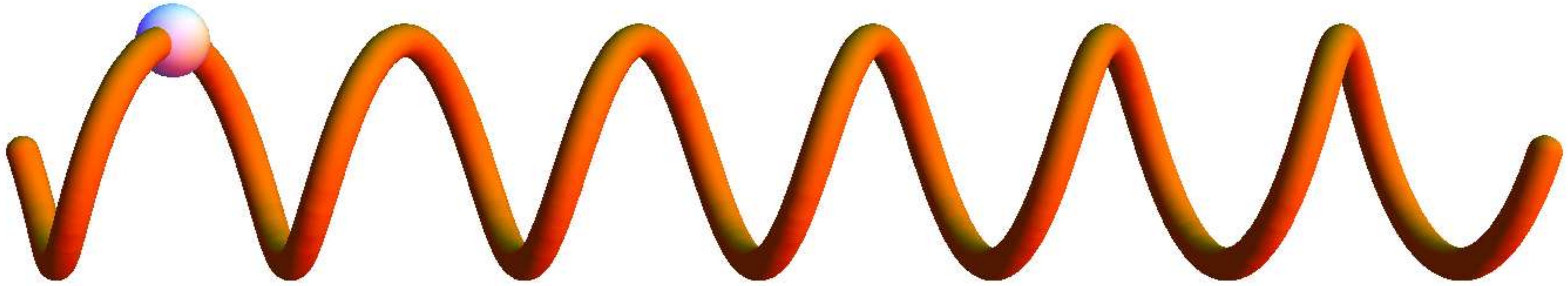


We already do it

- java.lang.String
- Effective Java

What's the point?

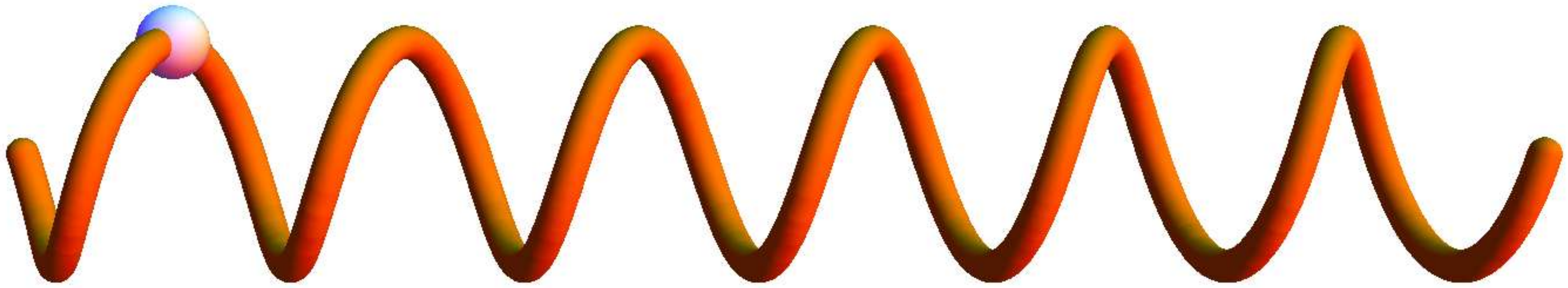
- The less state that can change, the less you have to think about.
- Concurrency!



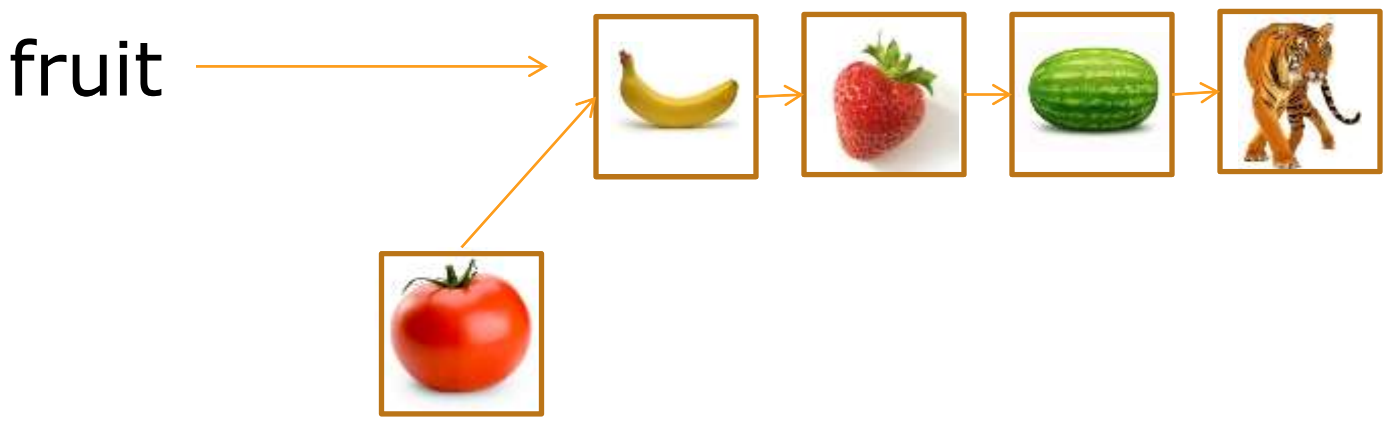
Immutability in functional languages

Pure: everything is immutable.

Hybrid: immutable by default



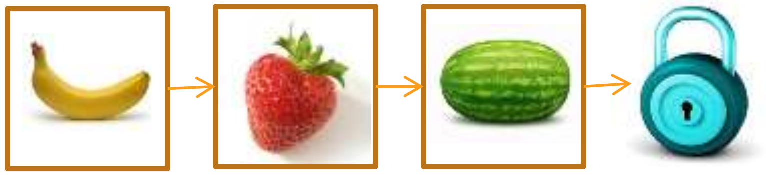
Immutability in functional languages



tomato :: fruit

Immutability in Java

final



Immutability in Java

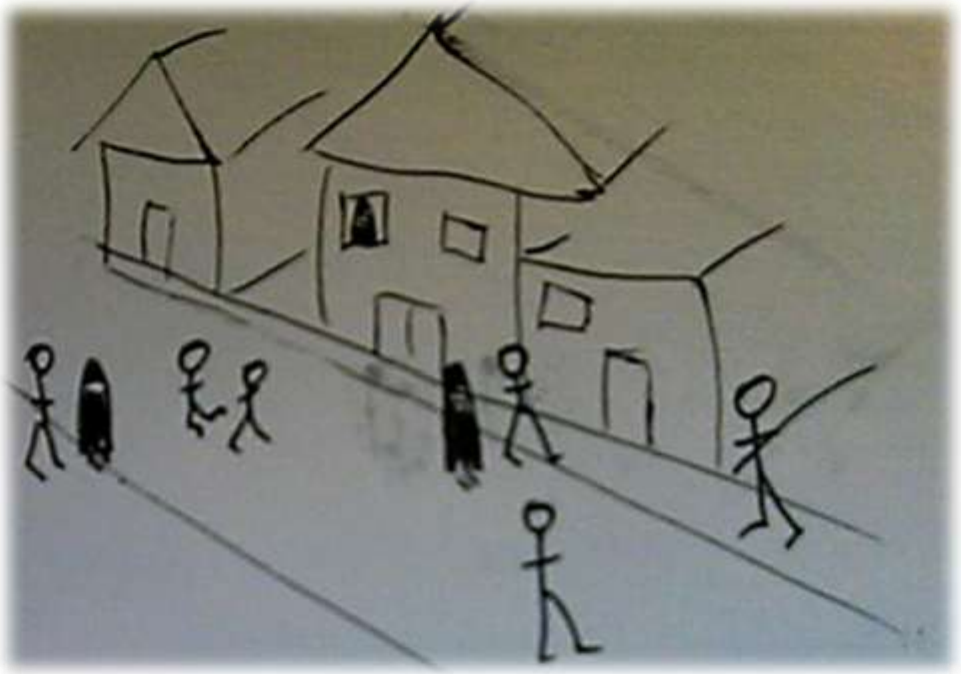
ImmutableMap.copyOf(mutableMap)

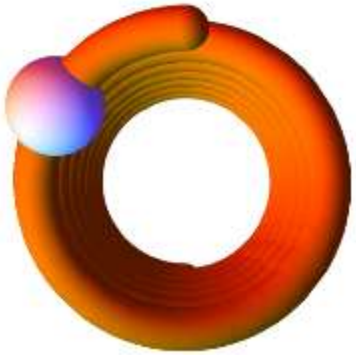
ImmutableList.of(item, item, item)

Immutability in Java

Keep it simple.

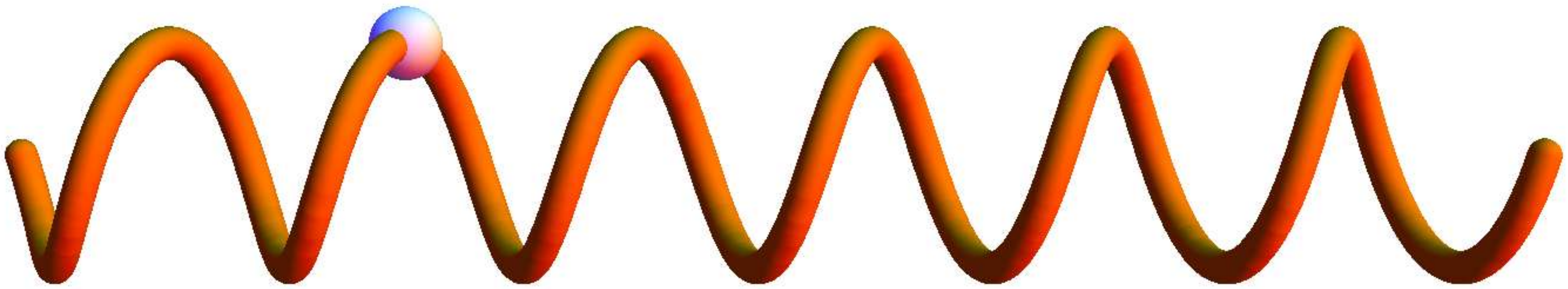
Verbs are people too





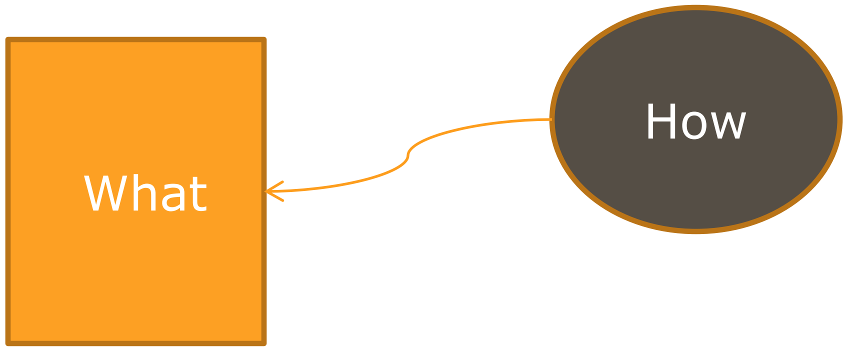
What is it?

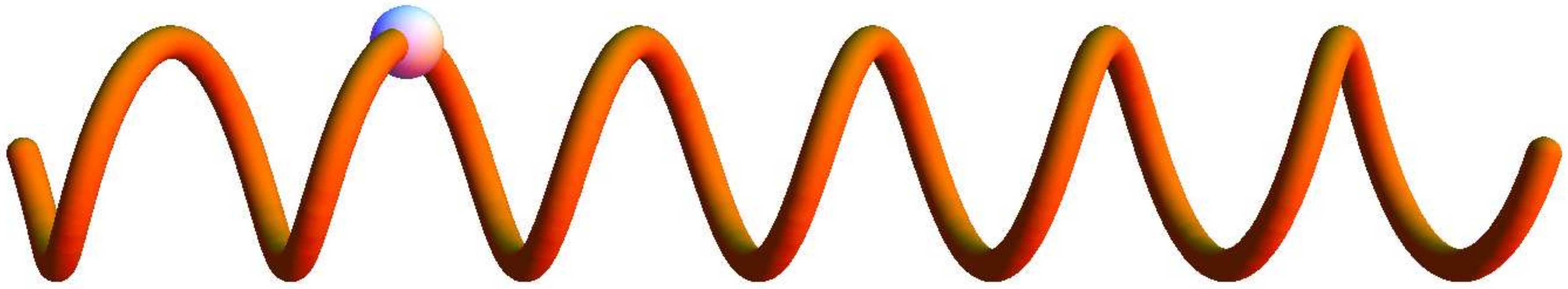
Functions are values. They can be passed around just like data.



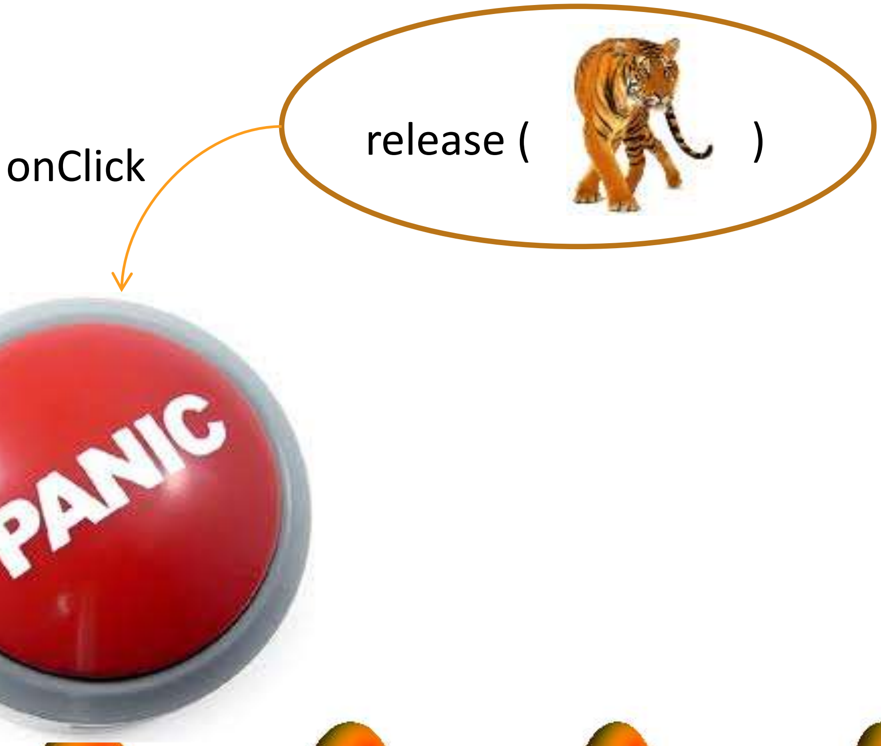
We already do it

- Strategy pattern
- Command pattern



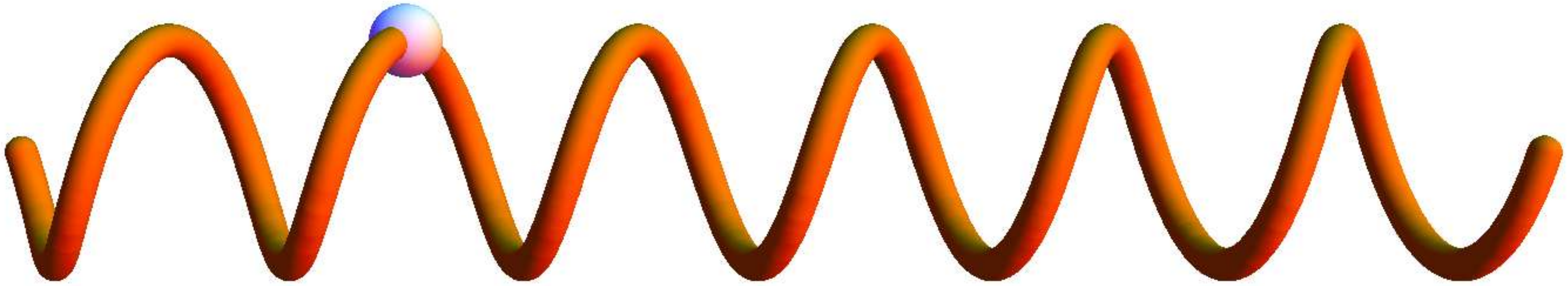


We already do it



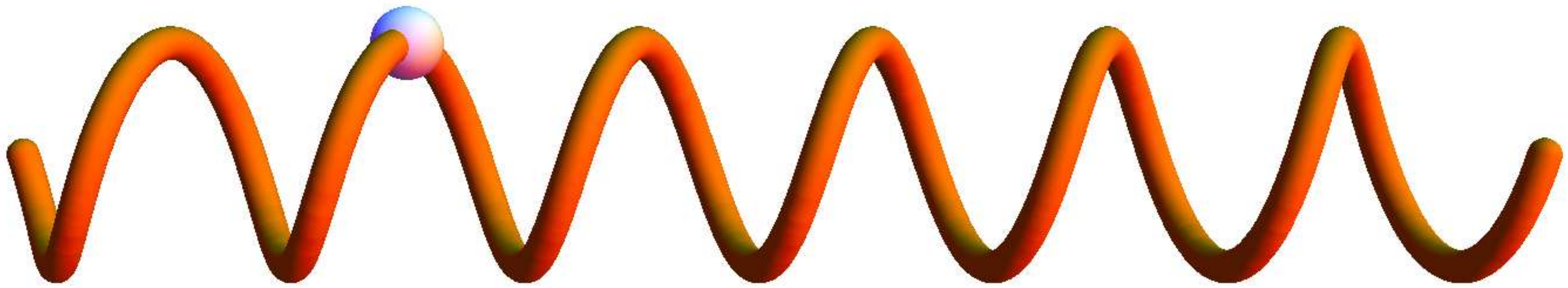
What’s the point?

Passing around instructions is useful.



Verbs in functional languages

```
case class User(val firstName : String)

val sortedList = userList.sortBy(u -> u.firstName)



 def getFirstName (u : User) = u.firstName
```

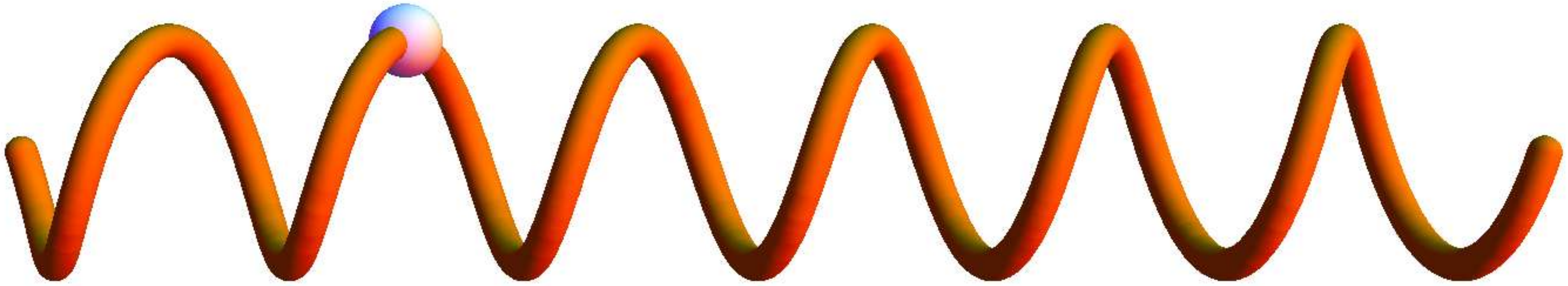


Verbs in Java

```
Collections.sort(myUserList,
  new Comparator<User>() {
    public int compare(User o1, User o2) {
      return
              o1.firstName.compareTo(o2.firstName);
    }
  };
);
```

Verbs in Java

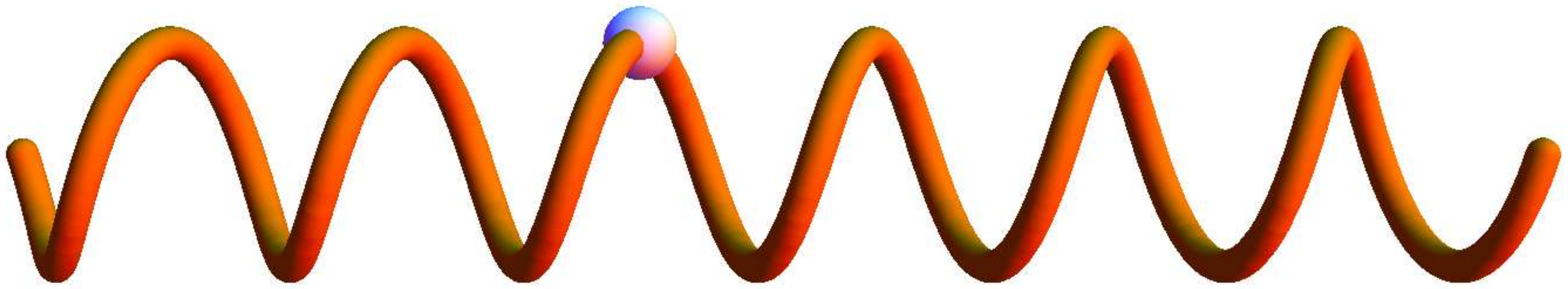
```
getFirstName = new Function<User, String>() {
    public String apply(User user) {
            return user.firstName;
        }
};

Ordering<User> o = Ordering.natural().onResultOf(getFirstName);
List<User> sortedList = o.sortedCopy(userList);
```

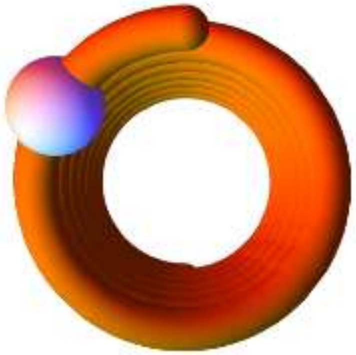


Verbs in Java 8

```
userList.sort(comparing(u -> u.firstName));

userList.sort(
        comparing(u -> u.firstName).reverseOrder()
    );
```

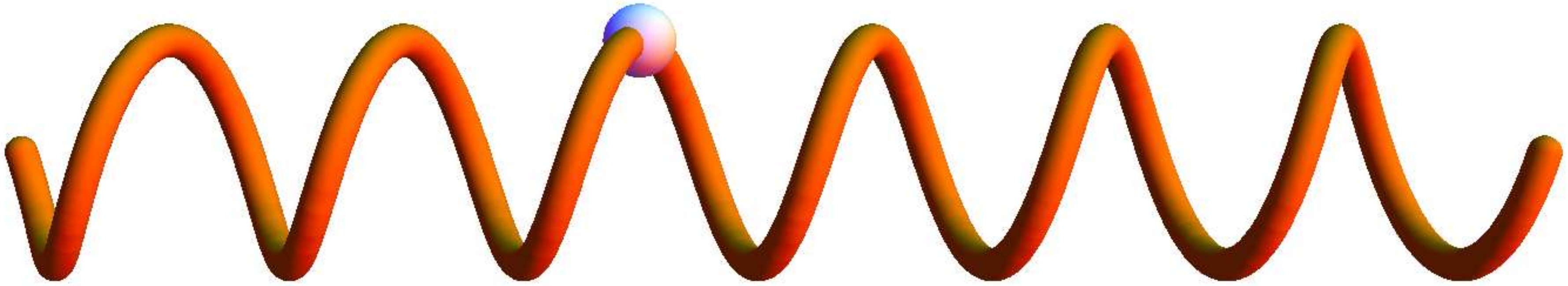
Declarative style





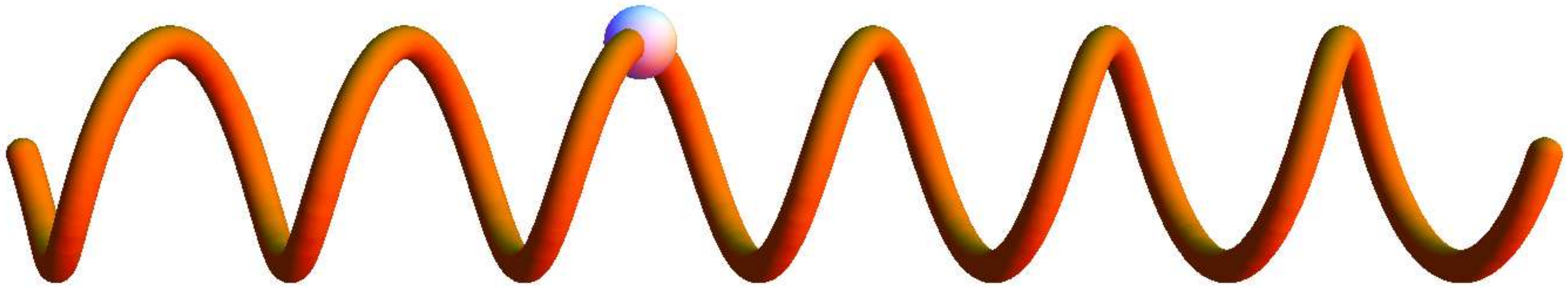
What is it?

Say what you’re doing, not how you’re doing it.



We already do it

- Refactoring: single-line method



We already do it

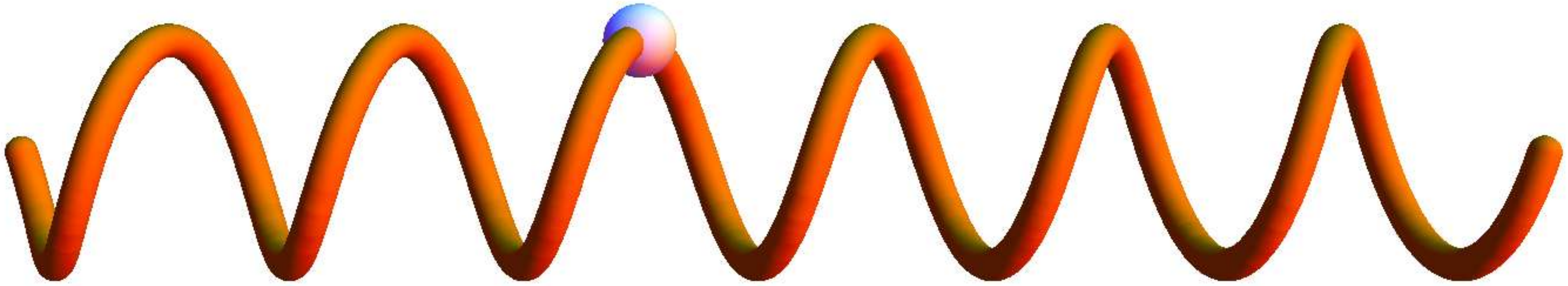
```
Select USER_NAME, count(*),
    max(update_date)
From USER_ROLES
Where USER_ID = :userId
Group by USER_NAME
```

What’s the point?

Readable code

Smaller, simpler pieces

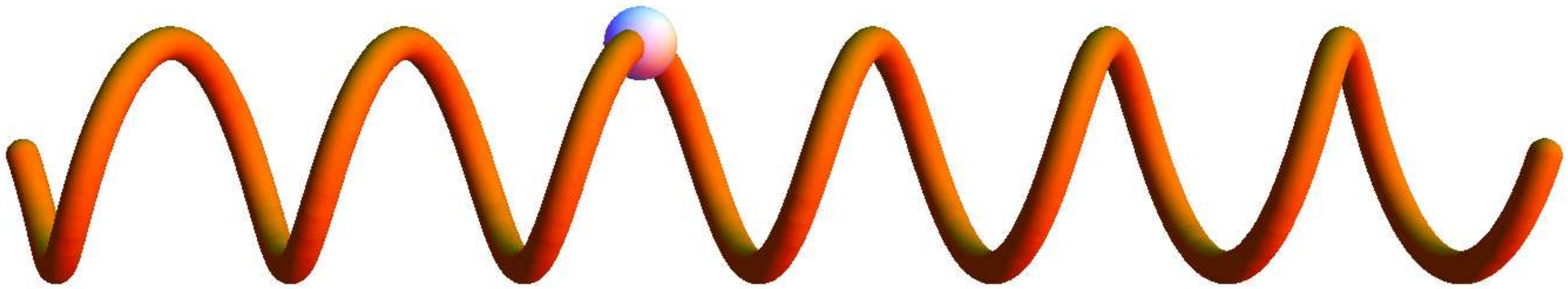
Familiar != readable



Declarative style in functional languages

- Many small functions
- One-line collection processing

```
linesFromFile.filter ( _.startsWith("BUG"))
```

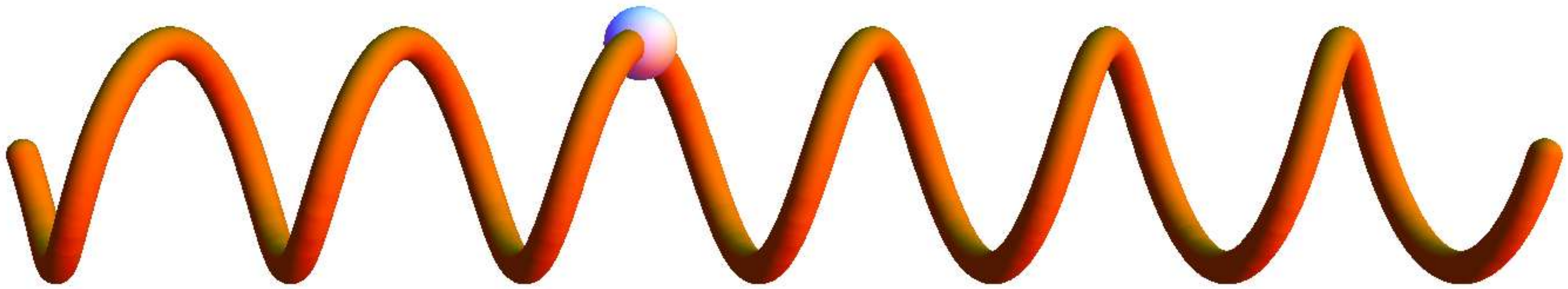


Declarative style in Java?

```
for (String line : list) {
    if (line.startsWith("BUG")) {
      report(line);
    }
}
```

Declarative style in Java

```
reportAll(filterForBugs(list));
```



Declarative style in Java?

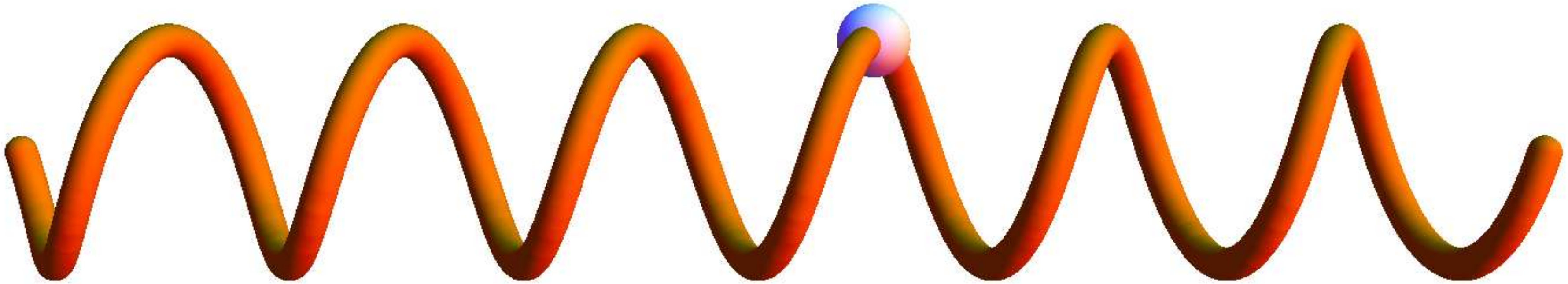
```
List<String> bugLines = new LinkedList<String>();
for (String line : list) {
    if (line.startsWith("BUG")) {
      bugLines.add(line);
    }
 }
return bugLines;
```

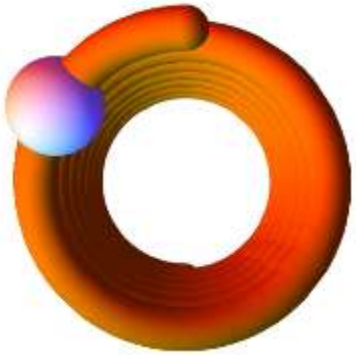
Declarative style in Java

```
Iterable<String> bugLines = filter(list, startsWithBug);
```

```
final Predicate<String> startsWithBug =
    new Predicate<String>() {
      public boolean apply(String s) {
        return s.startsWith("BUG");
      }
    };
```

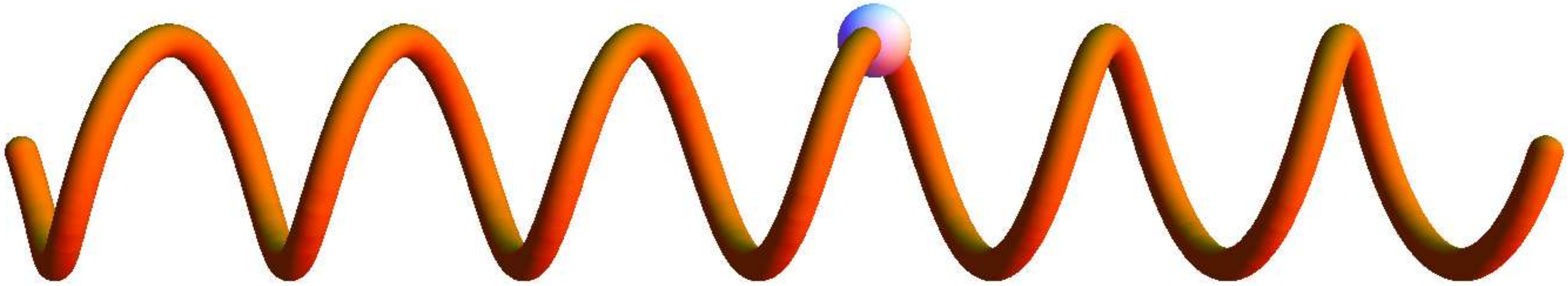
Null Is Your Enemy



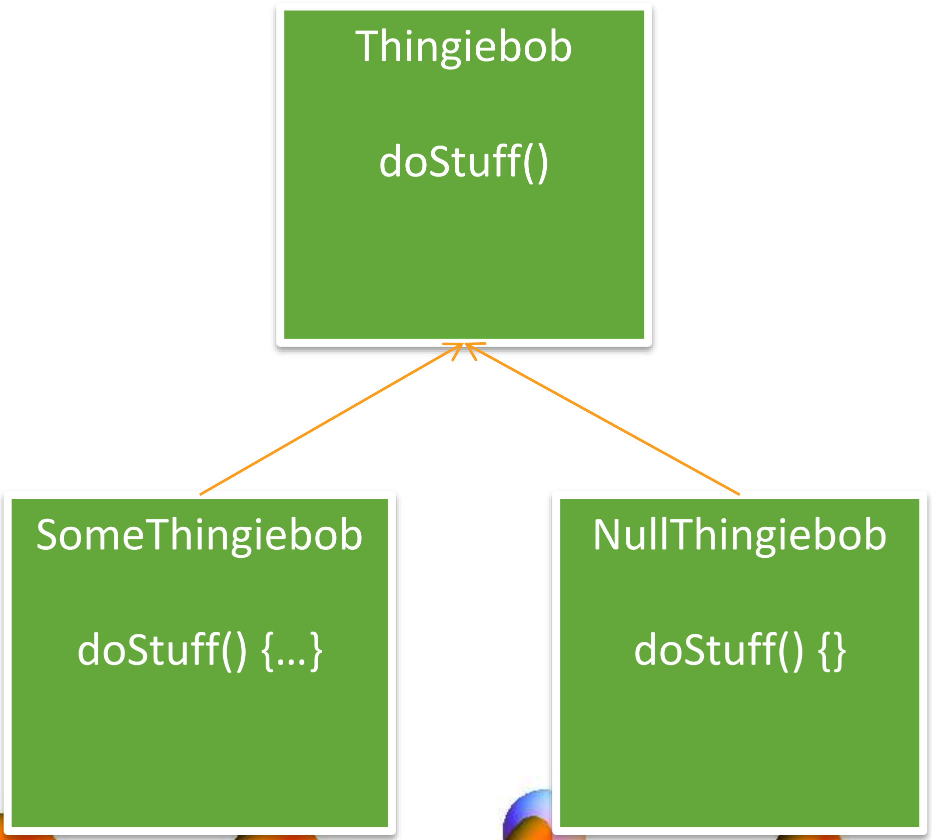


What is it?

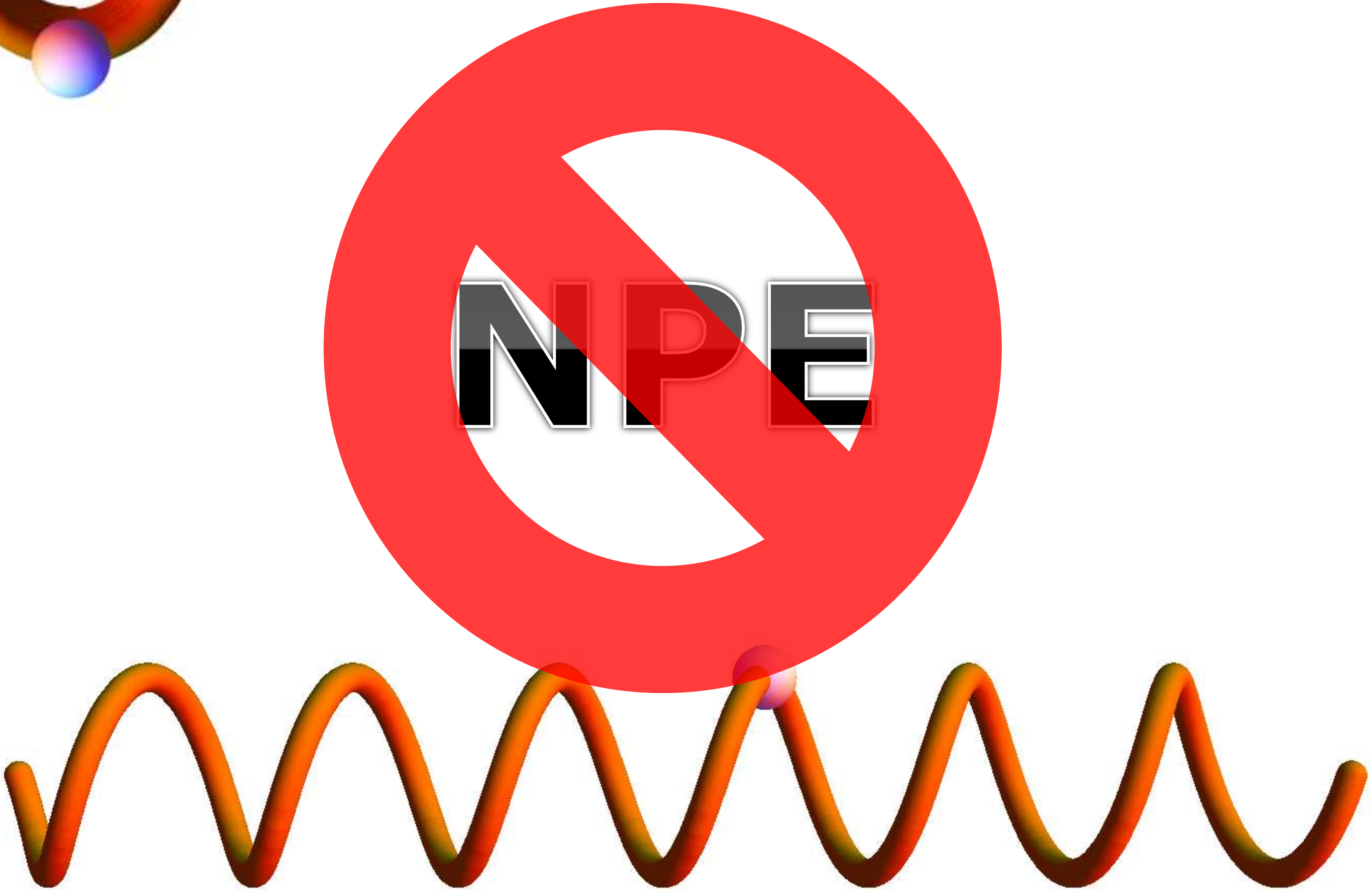
A null reference is not a valid object reference. Let’s stop treating it like one.



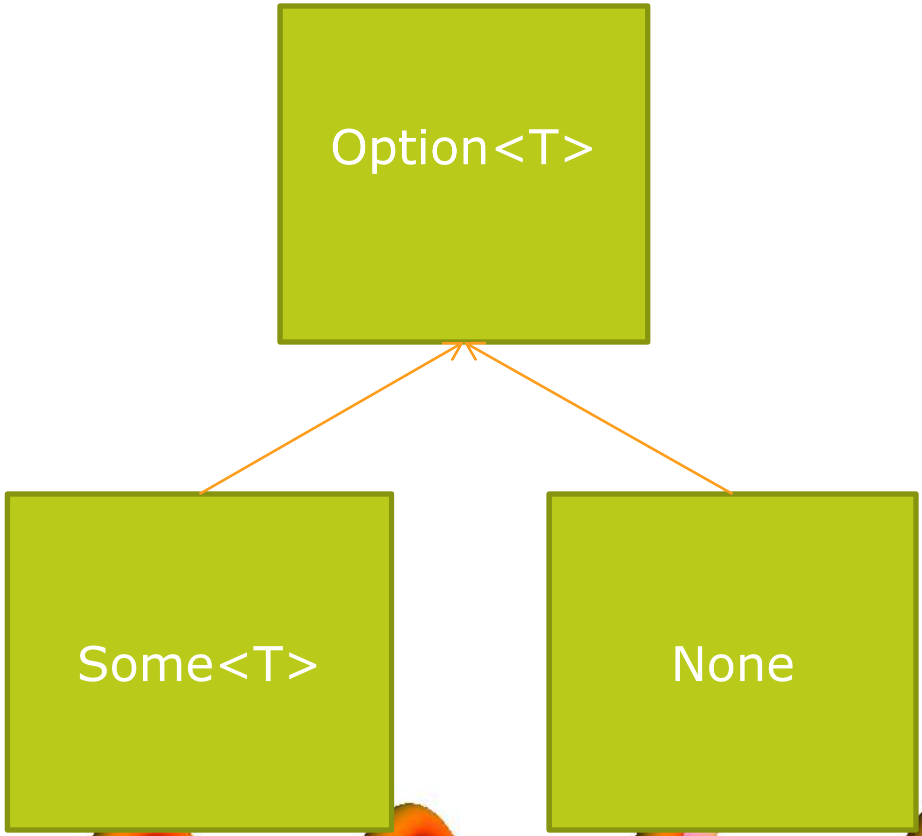
We already do it



What's the point?



Defeating null in functional languages



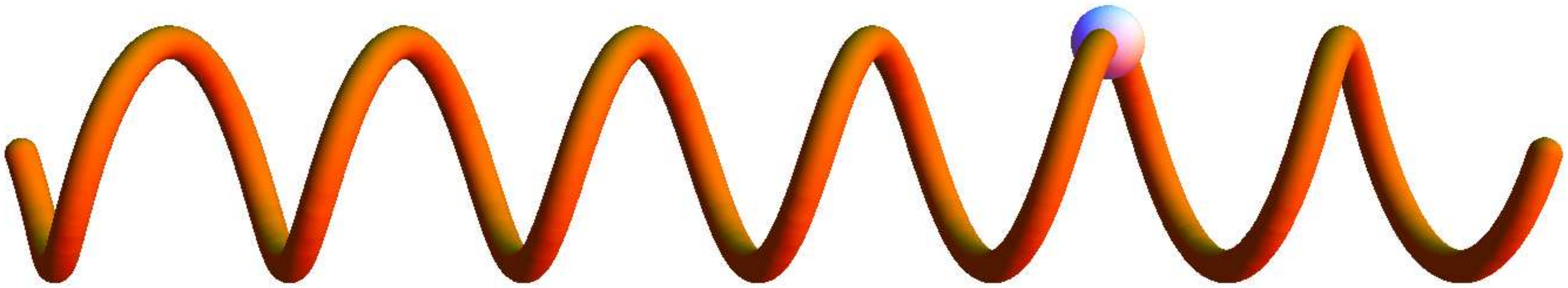
Defeating null in Java

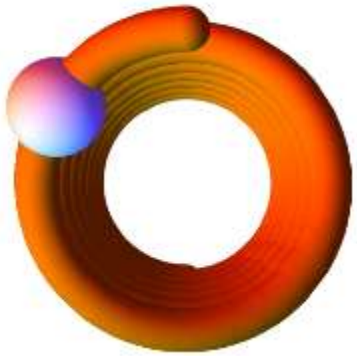
```
Optional<String> banana = Optional.of("banana");
Optional<String> noBanana = Optional.absent();


if (banana.isPresent()) {
   String contents = banana.get();
}
```

Defeating null in Java

```
Optional.fromNullable(mightBeNull);
```

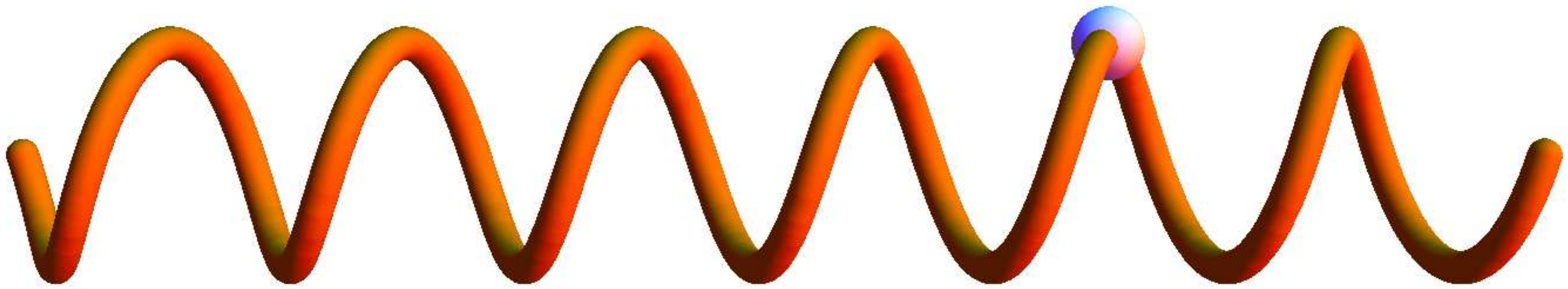
Strong typing





What is it?

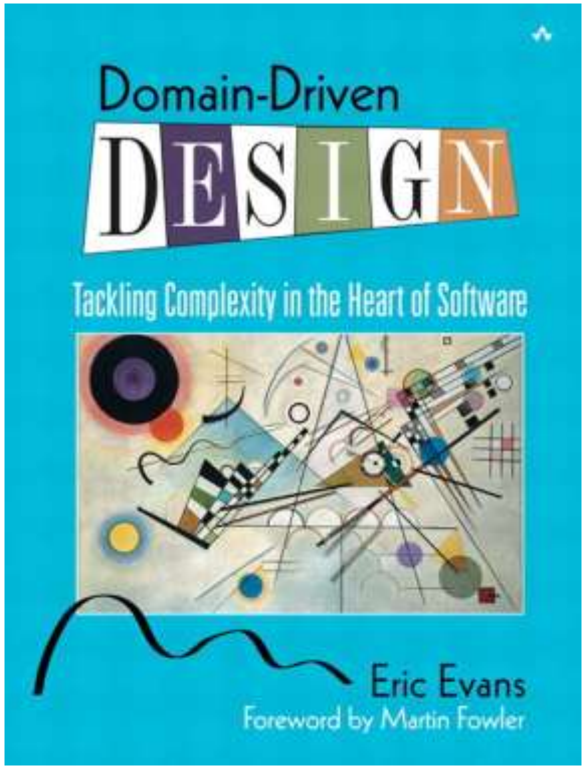
When the wrong type of data is passed in, the compiler complains.



We already do it

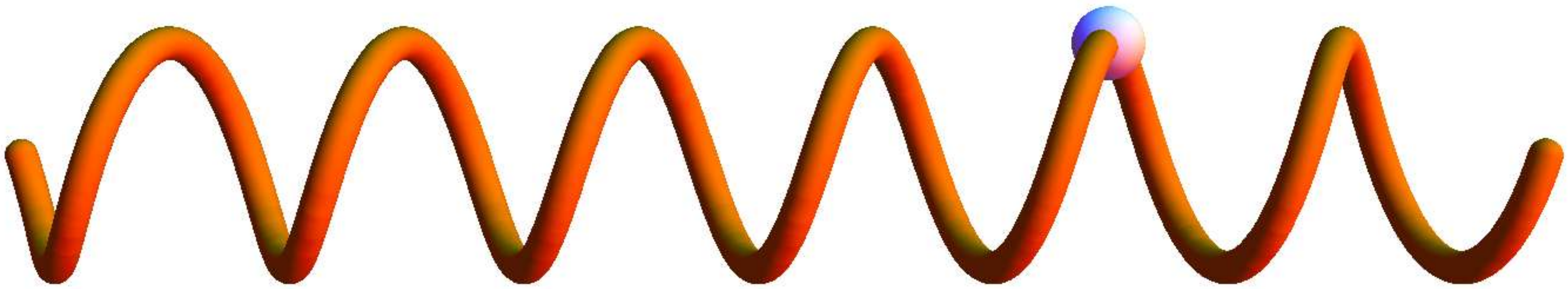
Java is strongly typed, right?

We already do it

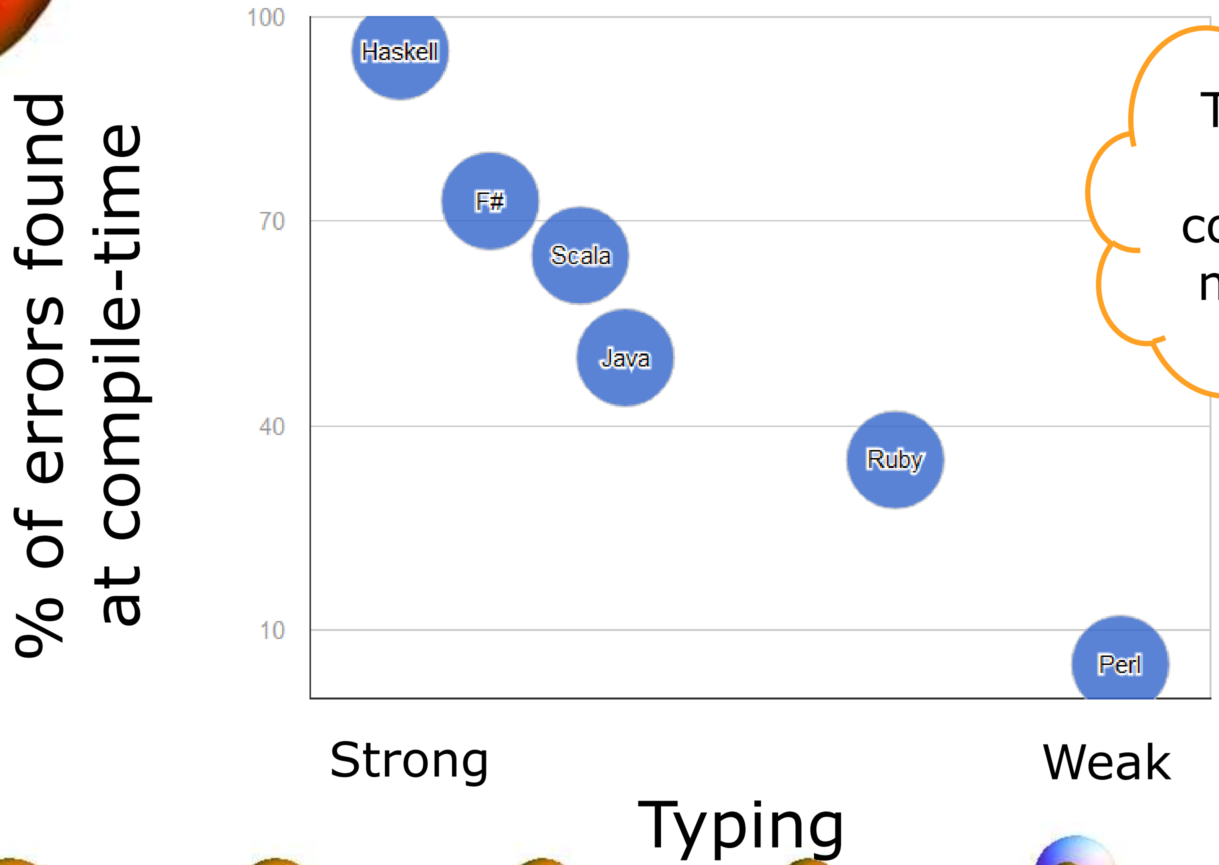

Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler

What’s the point?

The beginning of wisdom is to call things by their right names.



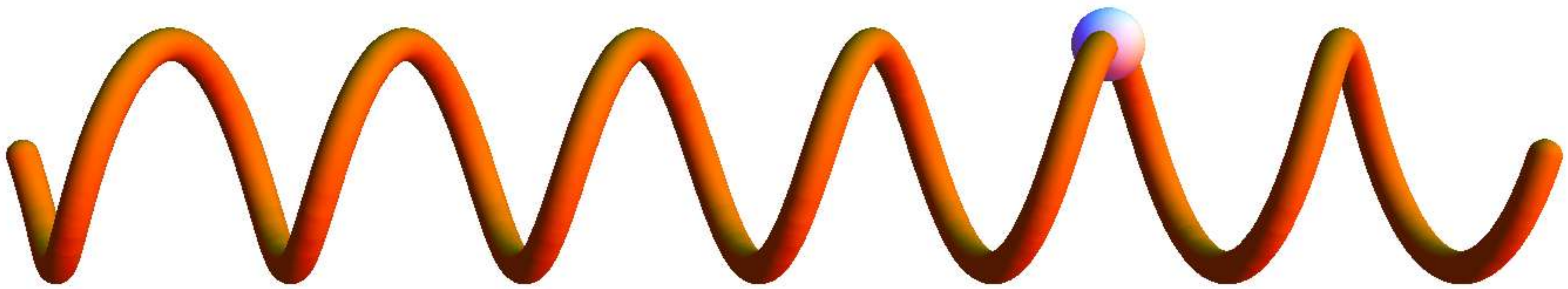
What's the point?



This data is completely made up.

Strong typing in functional languages

```
type FirstName = String              // Haskell type alias

data User = User FirstName EmailAddress
                                     // Haskell data type
```

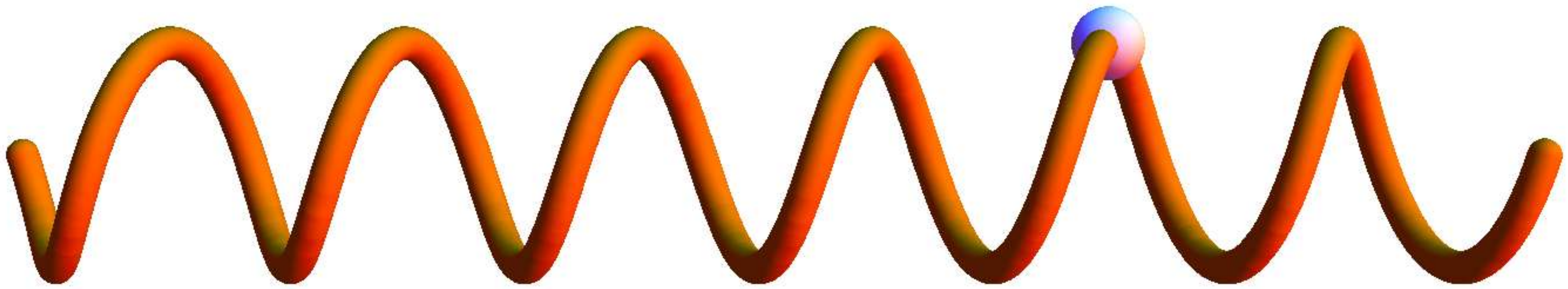


Strong typing in functional languages

List [+A]   // from Scaladoc

def **indexOf [B >: A] (elem: B): Int**

def
sameElements (that: GenIterable[A]): Boolean



Strong typing in Java

```
public class FirstName {
   public final String stringValue;

   public FirstName(final String
value) {
      this.stringValue = value;
   }

   public String toString() {...}
   public boolean equals() {...}
   public int hashCode() {...}
}
```

```
public User(FirstName name, EmailAddress login)
```

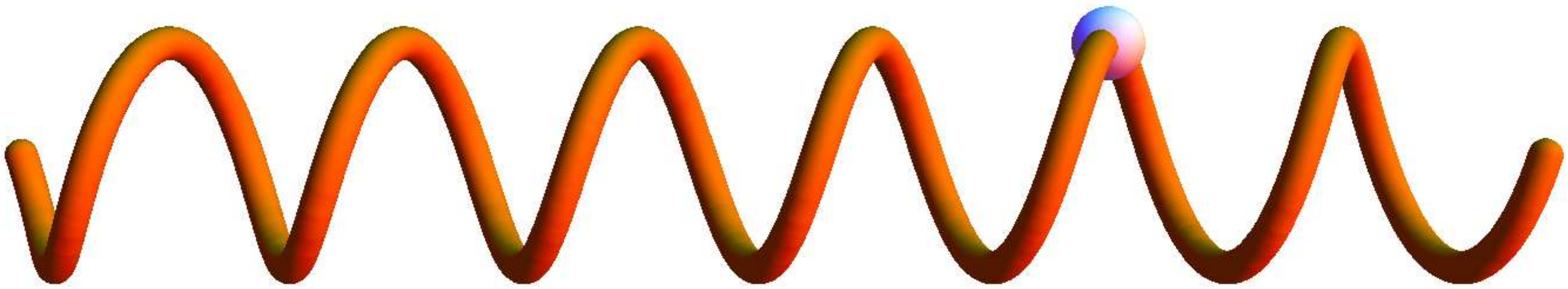
Strong typing in Java

```
new User(firstName(“Joe”),
                emailAddress(“joe@gmail.com”));
```

```
public static FirstName firstName(String value)
{
  return new FirstName(value);
}
```

Strong typing in Java

```
public boolean validateUser(User user)
{
   EmailAddress email = user.getEmailAddress();
   // exercise business logic
   return true;
}
```

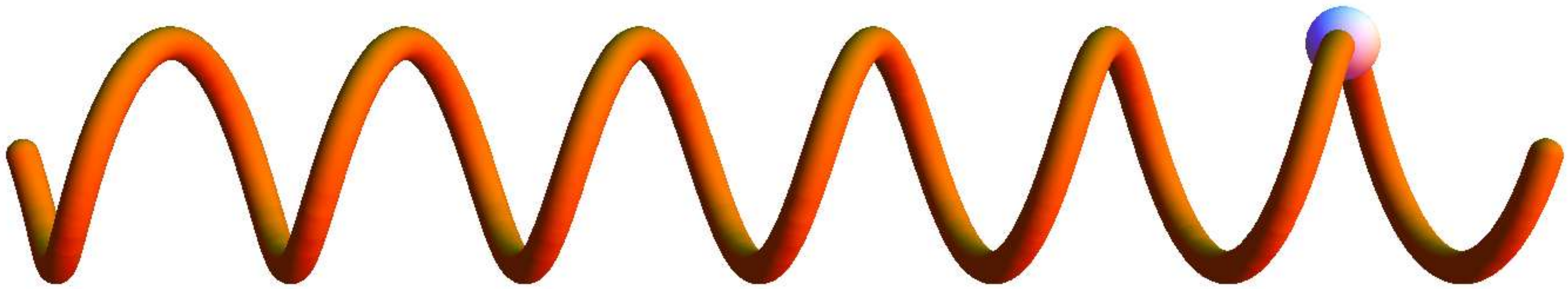


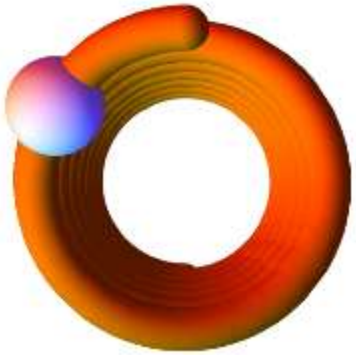
Strong typing in Java

```
public boolean validate(HasEmailAddress anything)
{
   EmailAddress email = anything.getEmailAddress();
   // exercise business logic
   return true;
 }
```

```
interface HasEmailAddress {
   EmailAddress getEmailAddress();
 }
```

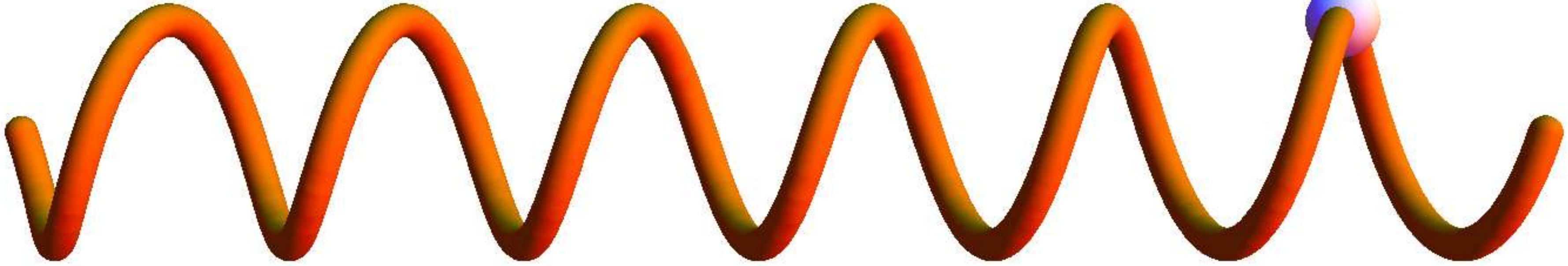
Lazy evaluation





What is it?

Delaying evaluation of an expression until the last responsible moment.



We already do it

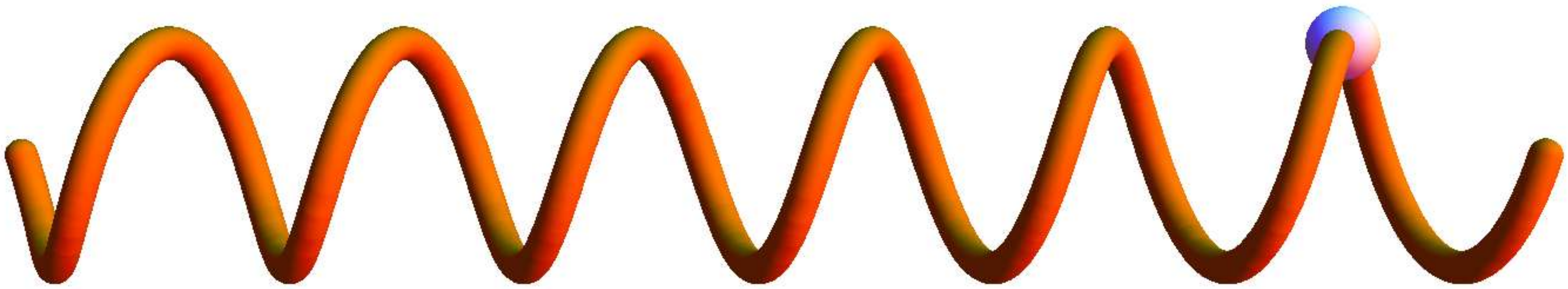
Providers, Factories

SQL Cursors

What’s the point?

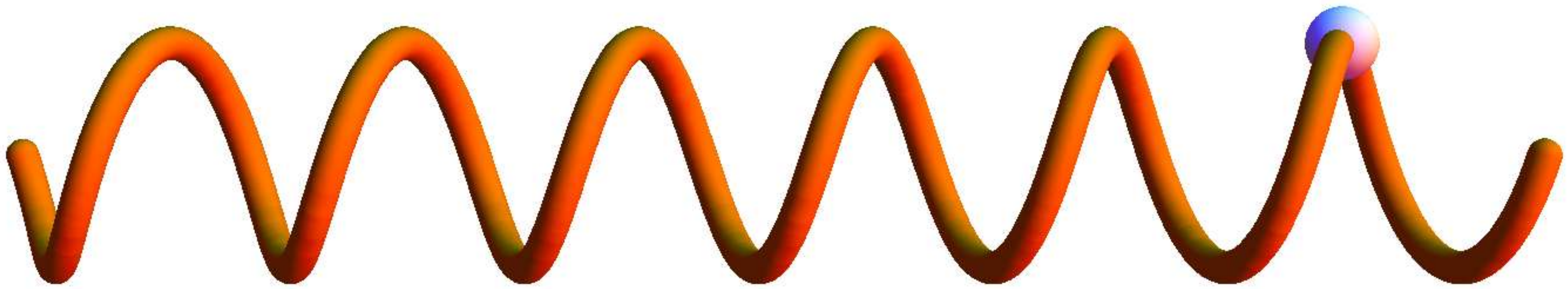
You may never even need it.

Separate “what to do” from “when to stop.”



Lazy evaluation in functional languages

- Haskell is lazy by default
- F# provides a Lazy<_> type
- Infinite sequences



Lazy evaluation in Java

Callable

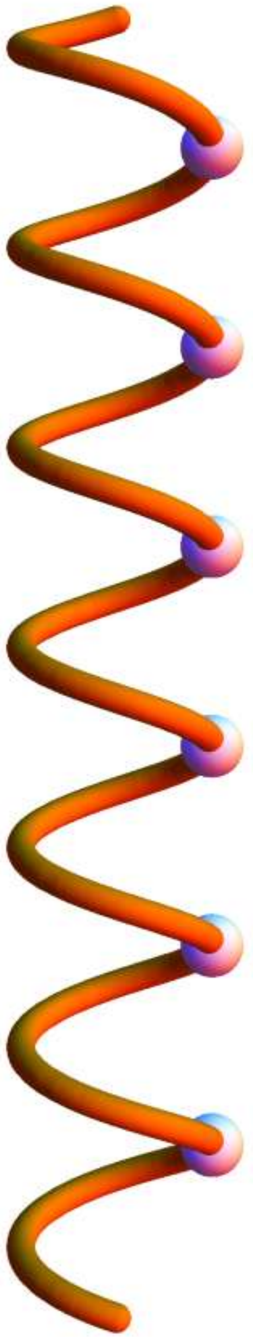
Iterable

Imperative Java

```
int bugCount = 0;
String nextLine = file.readLine();
while (bugCount < 40) {
  if (nextLine.startsWith("BUG")) {
    String[] words = nextLine.split(" ");
    report("Saw the bug at "+words[0]+" on "+ words[1]);
    bugCount++;
  }
  waitUntilFileHasMoreData(file);
  nextLine = file.readLine();
}
```

Functional style

```
for (String s : take(new RandomFileIterable(br))
        .filterBy(STARTS_WITH_BUG_PREDICATE)
        .transformWith(TRANSFORM_BUG_FUNCTION)
        .limit(40)
        .asImmutableList()) {
    report(s);
}
```



Immutability

Verbs Are People Too

Declarative Style

Null Is Your Enemy

Strong Typing

Lazy Evaluation

Thank you

Jessica Kerr @jessitron

Jessitron.blogspot.com

jessitron@gmail.com

http://speakerdeck.com/u/jessitron/

Look for me at KCDC

April 27-28 2012: kcdc.info