Beyond Java

자바 8을 중심으로 본 자바의 혁신

박성철(fupfin)

노땅 개발자(라고 착각하는 관리자)
SK Platnet CDP개발팀
한국 스프링 사용자 모임

~~말해도 모를 회사의 대표였지만 망했어요~~

설득력 없는 설득을 하는 사람이 있습니다.

자바 8을 중심으로 본 자바의 혁신

자바 8을 중심으로 본 자바의 혁신

β
1995

1.0
1996

1.1
1997

1.2
1998

1.3
2000

1.4
2002

5
2004

6
2006

8
2014

7
2011

나즈혼
자바스크립트 엔진

람다

기본 메서드

스트림 API

날짜 API

동시성 개선

Base64

메타프로그래밍 향상


절망했다!!!!


H
만남은 항상 경건하게
조용히 맞이해야 돼요

나즈혼
자바스크립트 엔진

람다

기본 메서드

스트림 API

날짜 API

동시성 개선

Base64

메타프로그래밍 향상

람다식

- 람다식은 메서드와 유사
- 형식 매개변수의 목록과
- 이 매개변수로 표현된 본문(식이나 코드 블럭)를 제공

A lambda expression is like a method: it provides a list of formal parameters and a body—an expression or block—expressed in terms of those parameters.

– jsr335, /spec/B.html

람다식

- 2006, 고슬링 "자바에 클로저가 들어오지는 않을 거야"
- 2007, 세 가지 다른 클로저 제안 제출
- 2008, 마크 라인홀드 "자바 7에 클로저는 안 들어가요"

  http://java.dzone.com/articles/java-7-update-mark-reinhold-de
- 2009, 람다 프로젝트(JSR 335) 시작

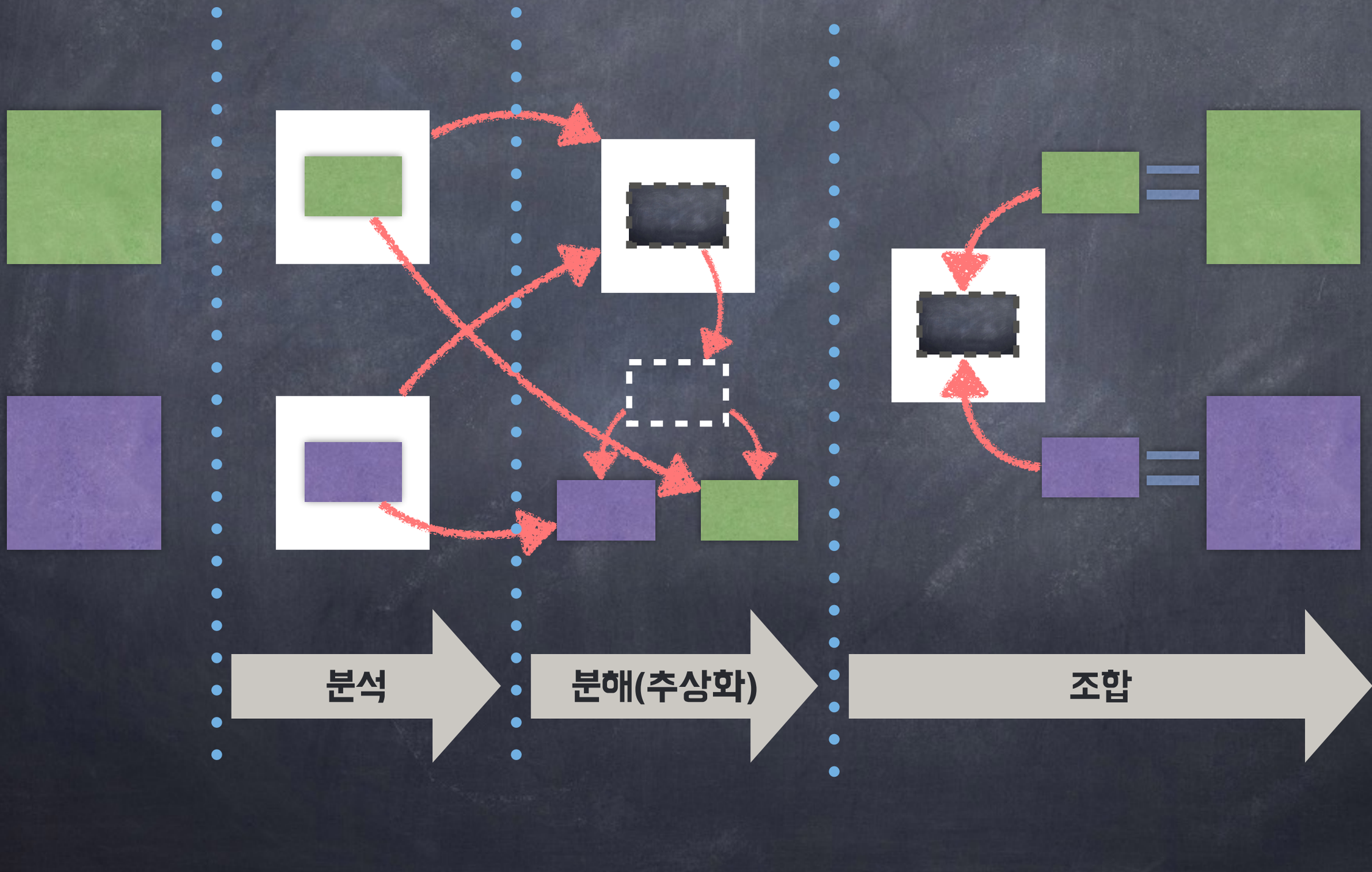
제어 흐름 중복

```
assert numbers.length > 0;

int max = numbers[0];
for(int i=1;
        i < numbers.length;
        i++)
    if(max < numbers[i])
        max = numbers[i];
return max;
```

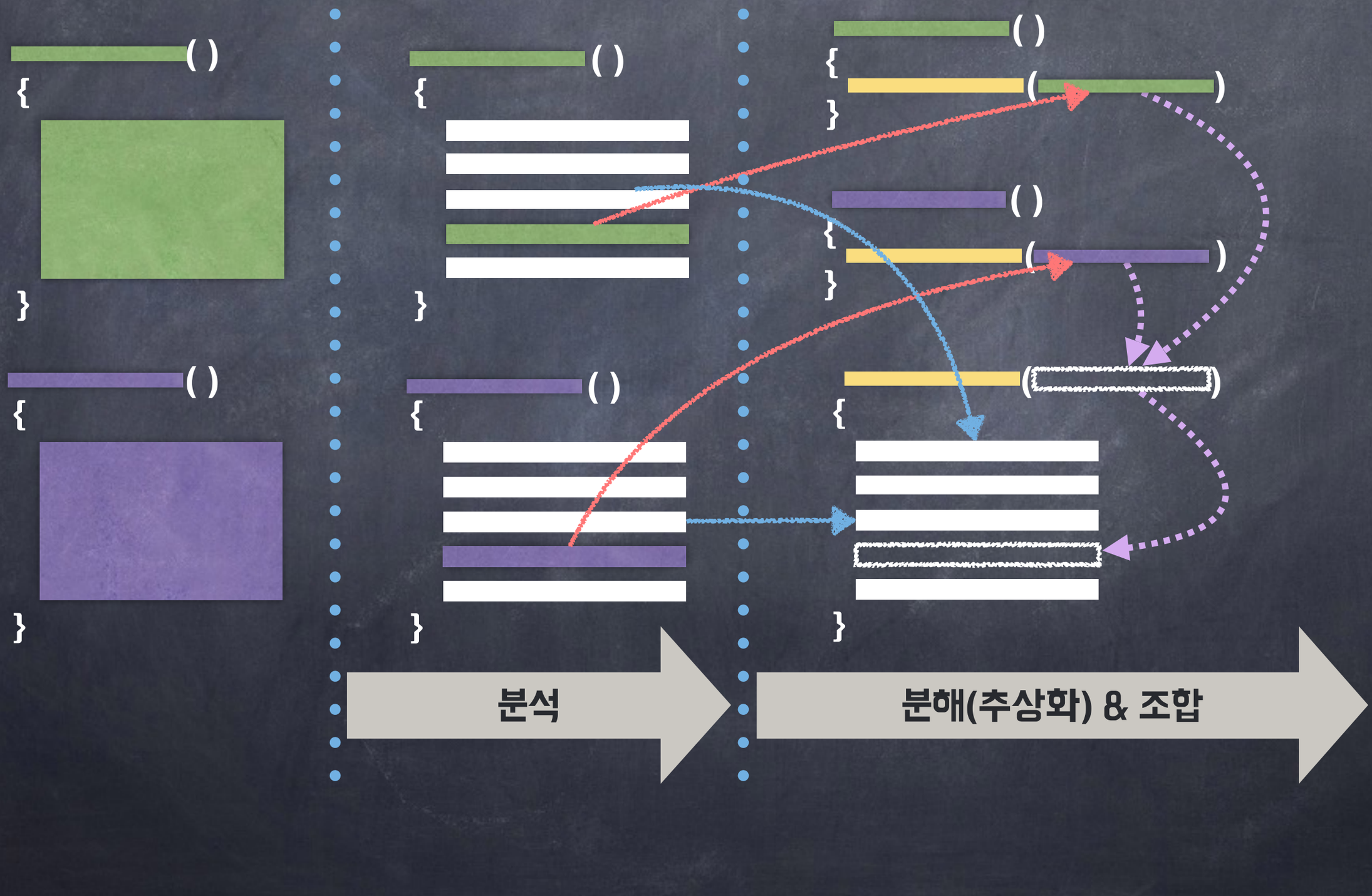
```
assert numbers.length > 0;

int max = numbers[0];
for(int i=1;
        i < numbers.length;
        i++)
    if(Math.abs(max) <
       Math.abs(numbers[i])
        max = numbers[i];
return max;
```

DRY: Don't Repeat Yourself!!!

전략 패턴



행위 매개변수



자바의 행위 매개변수 예

자바에서는 객체를 사용해서 행위를 매개변수로 전달

```
java.util.Collections

public static <T>
T max(Collection<? extends T> coll, Comparator<? super T> comp)

public static <T>
T min(Collection<? extends T> coll, Comparator<? super T> comp)

public static <T>
void sort(List<T> list, Comparator<? super T> c)
```

자바의 행위 매개변수 예
(익명 클래스)

java.util.Collections

```
public static <T> void sort(List<T> list, Comparator<? super T> c)
```

```
Collections.sort(items, new Comparator<Item>() {
    @Override
    public int compare(Item item1, Item item2) {
        return item1.getWeight() - item2.getWeight();
    }
});

Collections.sort(members, new Comparator<Member>() {
    @Override
    public int compare(Member member1, Member member2) {
        return member1.getAge() - member2.getAge();
    }
});
```

행위 매개변수 용도

로직 조합 = 전략 패턴(Strategy Pattern)

```
Arrays.sort(items, new Comparator<Item>() {
    @Override
    public int compare(Item item1, Item item2) {
        return item1.getWeight() - item2.getWeight();
    }
});
```

콜백 전달 = 관찰자 패턴(Observer Pattern)

```
button.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        doSomeAwesomeTask();
    }
});
```

익명 클래스의 문제

매번 객체 생성

클래스 선언

```
Arrays.sort(items, new Comparator<Item>() {
    @Override
    public int compare(Item item1, Item item2) {
        return item1.getWeight() - item2.getWeight();
    }
});
```

정작 필요한 부분

불연속적 구분 범위(Lexical Scope)

람다식(익명 메서드) 문법

(인자 목록) -> { 구문 }

```
Arrays.sort(items, new Comparator<Item>() {
    @Override
    public int compare(Item item1, Item item2) {
        return item1.getWeight() - item2.getWeight();
    }
});

Arrays.sort(items, (item1, item2) -> {
                        return item1.getWeight() - item2.getWeight()
                    });

Arrays.sort(items, (item1, item2) -> item1.getWeight() - item2.getWeight());
```

함수 인터페이스

- 추상 메서드가 하나인 인터페이스
- 단일 함수(기능) 규약을 기술
- Object 클래스의 메서드는 제외

A functional interface is an interface that has just one abstract method (aside from the methods of Object), and thus represents a single function contract.

– jsr335, /spec/A.html

람다 -> 함수 인터페이스

람다식으로 표현된 식이나 코드는 함수 인터페이스의 구현체로 변환됨

호출하는 측

```
Arrays.sort(items, (item1, item2) -> item1.getWeight() - item2.getWeight());
```

변환

호출받는 측

```
public static <T> void sort(T[] a, Comparator<? super T> c){
    ...
      if(c.compare(o1, o2)) {
          ...
      }
    ...
}
```

람다식의 예

타입 추론

구문 블럭

식

단일 인자 괄호 생략

빈 인자

```
(int n, String str) -> { return str + n;}

(int n, String str) -> str + n

(n, str) -> str + n

str -> str + 1

() -> “Hello, World!”

() -> {}
```

람다의 필요성

- 다양한 언어에서 고차함수 지원 (C++ 11)
- C# 3.0 람다식 지원 & .Net 프레임워크(LINQ) (2007)
- 함수형 프로그래밍 보급 (멀티 코어 CPU 시대)
- 스프링 xxxTemplate
- 자바의 수다스러운 구문에 대한 반발: Guava, LambdaJ, Op4j

스프링 프레임워크의 JdbcTemplate 예

```
List<Actor> actors = this.jdbcTemplate.query(
        "select first_name, last_name from t_actor",
        new RowMapper<Actor>() {
            public Actor mapRow(ResultSet rs, int rowNum) throws SQLException {
                Actor actor = new Actor();
                actor.setFirstName(rs.getString("first_name"));
                actor.setLastName(rs.getString("last_name"));
                return actor;
            }
        });


List<Actor> actors = this.jdbcTemplate.query(
        "select first_name, last_name from t_actor",
        (rs, rowNum) -> {
                Actor actor = new Actor();
                actor.setFirstName(rs.getString("first_name"));
                actor.setLastName(rs.getString("last_name"));
                return actor;
            }
        });
```

나즈혼
자바스크립트 엔진

람다

기본 메서드

스트림 API

날짜 API

동시성 개선

Base64

메타프로그래밍 향상

컬랙션 람다 적용

- 구식 컬랙션(Java 1.2) API
- 컬랙션 2 제작? 너무 큰 작업, 너무 많은 레거시
- 교체 대신 **진화!**
- 기존 인터페이스 확장(기본 메서드)과 **스트림** 추상화

외부 반복 VS 내부 반복

명시적 제어 흐름

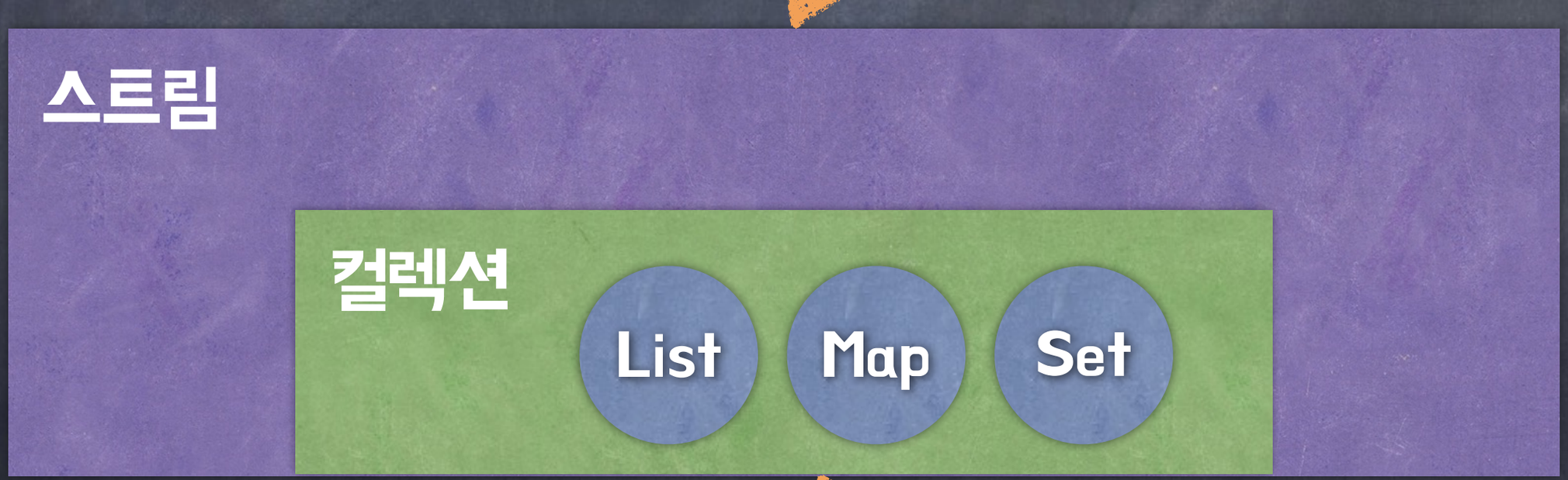
```
for (Shape s : shapes) {
    s.setColor(RED);
}
```

내부에 숨은 제어 흐름

```
shapes.forEach(s -> s.setColor(RED));
```

컬렉션과 스트림

처리 추상화 자료구조



보관용 자료구조

스트림 API

- java.util.stream
- 연산의 선언적 서술
  - 반복 구조 캡슐화, 알고리듬 분리
  - 제어 흐름 추상화
- 함수형 방식 처리: 데이터 불변
- 지연 처리, 대부분의 작업을 단일 반복 수행으로 처리
- 무한 연속 데이터 흐름 API
- 다양한 데이터 원천 지원: 컬렉션, 생성 메서드, 파일 I/O 등

스트림의 종류

Stream<T>	객체를 요소로 하는 가장 일반적인 스트림
DoubleStream	요소가 double형인 스트림
IntStream	요소가 int형인 스트림
LongStream	요소가 long형인 스트림

병렬처리

순차 처리 스트림

```
int sum = shapes.stream()
               .filter(s -> s.getColor() == BLUE)
               .mapToInt(s -> s.getWeight())
               .sum();
```

병렬 처리 스트림

```
int sum = shapes.parallelStream()
               .filter(s -> s.getColor() == BLUE)
               .mapToInt(s -> s.getWeight())
               .sum();
```

aParallelStream.sequential() ⟷ aSeqStream.parallel()

변환

스트림 적용 확산

***Stream**	`of(*), range(…)`
Collection	`stream(), parallelStream()`
BufferedReader	`lines()`
Random	`doubles(*), ints(*), longs(*)`
Arrays	`stream(*)`
***Stream.Builder**	`build()`
BitSet	`IntStream()`

비 스트림 forEach() 적용

```
Iterable.forEach(…) //*List, *Set, *Map, *Queue, *Deque, Vector 등등
Map.forEach(…)
```

jsr335-final/info/sotc.html

기본 메서드

- 인터페이스에 기본 구현 포함 가능
- 추가된 인터페이스의 하위 호환성 보장
- default 키워드 사용
- 함수형 인터페이스 조건에서 제외
- 인터페이스 확장시 재구현 또는 추상(abstract)으로 재지정 가능

```
public interface Iterator<E> {

    ......

    default void remove() {
        throw new UnsupportedOperationException("remove");
    }

    default void forEachRemaining(Consumer<? super E> action) {
        Objects.requireNonNull(action);
        while (hasNext())
            action.accept(next());
    }
```

스트림 사용 예

```
List<Album> favs = new ArrayList<>();
for (Album a : albums) {
    boolean hasFavorite = false;
    for (Track t : a.tracks) {
        if (t.rating >= 4) {
            hasFavorite = true;
            break;
        }
    }
    if (hasFavorite)
        favs.add(a);
}
Collections.sort(favs, new Comparator<Album>() {
                           public int compare(Album a1, Album a2) {
                               return a1.name.compareTo(a2.name);
                           }});
```

```
List<Album> sortedFavs =
  albums.stream()
        .filter(a -> a.tracks.anyMatch(t -> (t.rating >= 4)))
        .sorted(Comparator.comparing(a -> a.name))
        .collect(Collectors.toList());
```

jsr335-final/info/sotc.html

스트림 사용 예

```
for (Method m : enclosingInfo.getEnclosingClass().getDeclaredMethods()) {
    if (m.getName().equals(enclosingInfo.getName()) ) {
        Class<?>[] candidateParamClasses = m.getParameterTypes();
        if (candidateParamClasses.length == parameterClasses.length) {
            boolean matches = true;
            for(int i = 0; i < candidateParamClasses.length; i++) {
                if (!candidateParamClasses[i].equals(parameterClasses[i])) {
                    matches = false;
                    break;
                }
            }
            if (matches && m.getReturnType().equals(returnType)) return m;
        }
    }
}
throw new InternalError("Enclosing method not found");
```

```
return Arrays.stream(enclosingInfo.getEnclosingClass().getDeclaredMethods())
             .filter(m -> Objects.equals(m.getName(), enclosingInfo.getName())
             .filter(m -> Arrays.equals(m.getParameterTypes(), parameterClasses))
             .filter(m -> Objects.equals(m.getReturnType(), returnType))
             .findFirst()
             .orElseThrow(() -> new InternalError("Enclosing method not found");
```

jsr335-final/info/sotc.html

나즈혼
자바스크립트 엔진

람다

기본 메서드

스트림 API

날짜 API

동시성 개선

Base64

메타프로그래밍 향상

앳우드의 법칙

"자바스크립트로 작성될 수 있는 애플리케이션이라면, 언젠가 자바스크립트로 작성될 것이다.
Any application that can be written in JavaScript, will eventually be written in JavaScript."

- Jeff Atwood, The Principle of Least Power
http://blog.codinghorror.com/the-principle-of-least-power/

나즈혼(Nashorn)
자바스크립트 엔진

- 구형 리노(Rhino)를 대체하는 기본 자바스크립트 엔진
- ECMAScript 5.1(ECMA-262) 준수
- 자바 7의 invokedynamic 활용, 성능 향상
- 자바 객체 호출 가능
- javax.script(JSR 223) API 지원

나즈혼(Nashorn)
자바스크립트 엔진

터미널에서 jjs 명령을 사용해 실행

```
$ $JAVA_HOME/bin/jjs
jjs> print('Hello World');
```

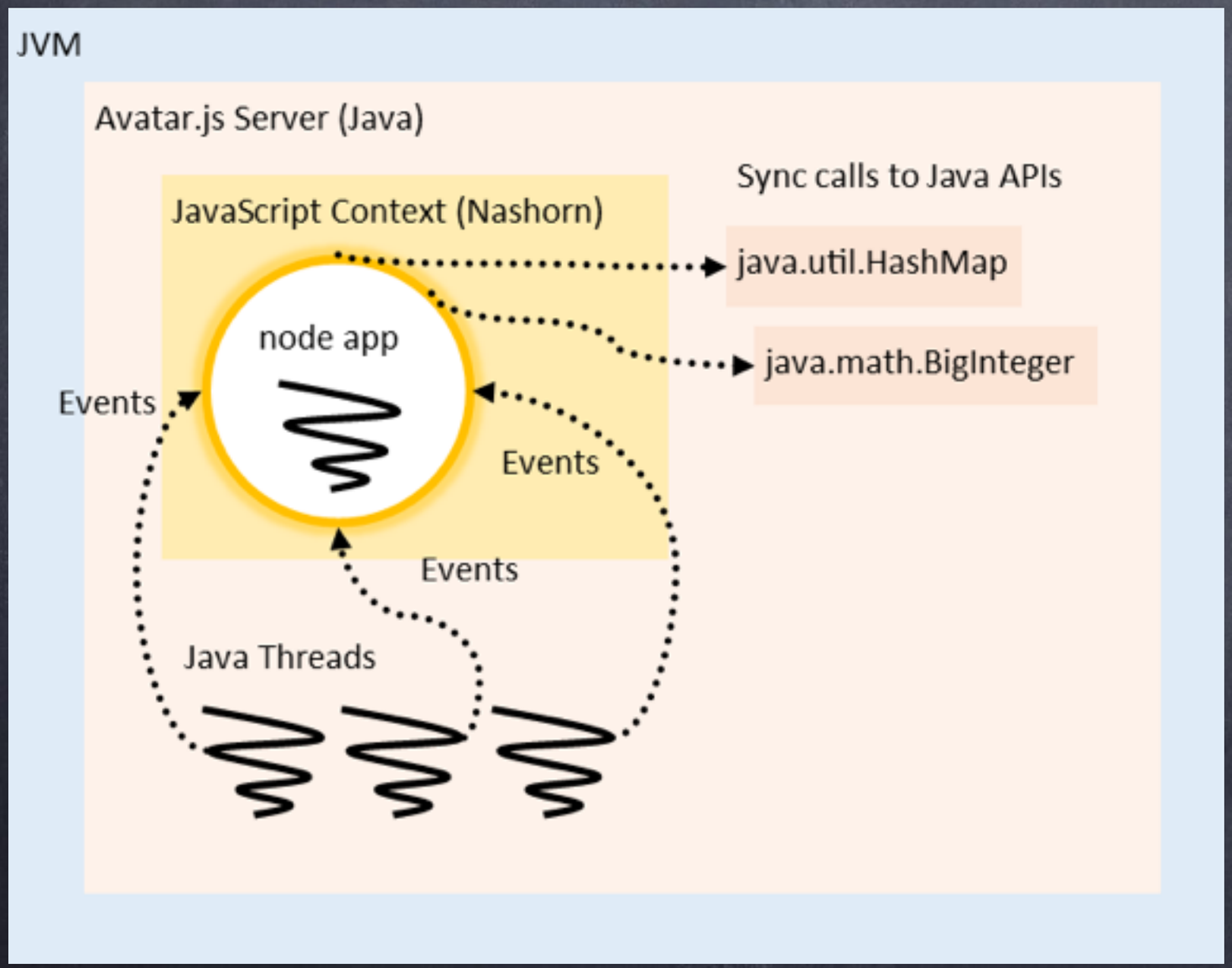
자바의 스크립트 엔진 매니저를 사용해 실행

```
ScriptEngine engine = new ScriptEngineManager().getEngineByName("nashorn");
engine.eval("print('Hello World!');");
```

성능 밴치마크

- Nashorn: The New Rhino on the Block: http://skpla.net/ggNQ
- Performance: Nashorn vs. Node : http://skpla.net/e9f4

Avatar.js
자바용 서버 자바스크립트



- Node.js 모델
- Node.js 호환
- 나스혼 엔진
- 자바 플랫폼
- 다중 쓰레드
- 다중 이벤트 루프

자바 8을 중심으로 본 자바의 혁신

β 1995

1.0 1996

1.1 1997

1.2 1998

1.3 2000

1.4 2002

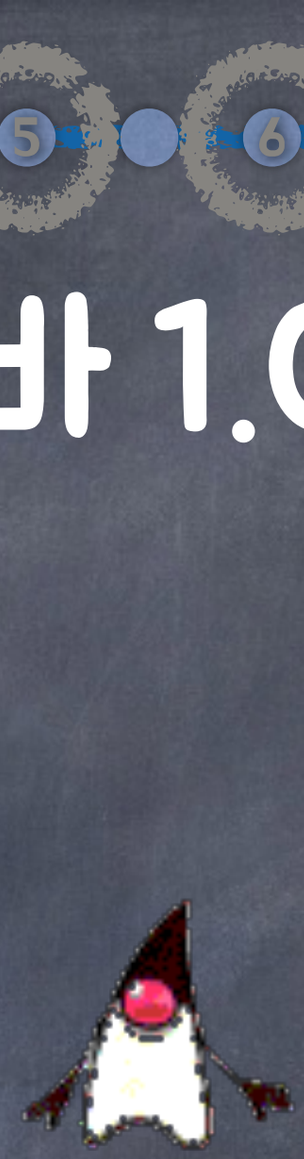
5 2004

6 2006

7 2011

8 2014

자바 1.0



자바 1.0

자바 백서에 따르면...

1. 단순함, 친숙함 = C++ - WTF

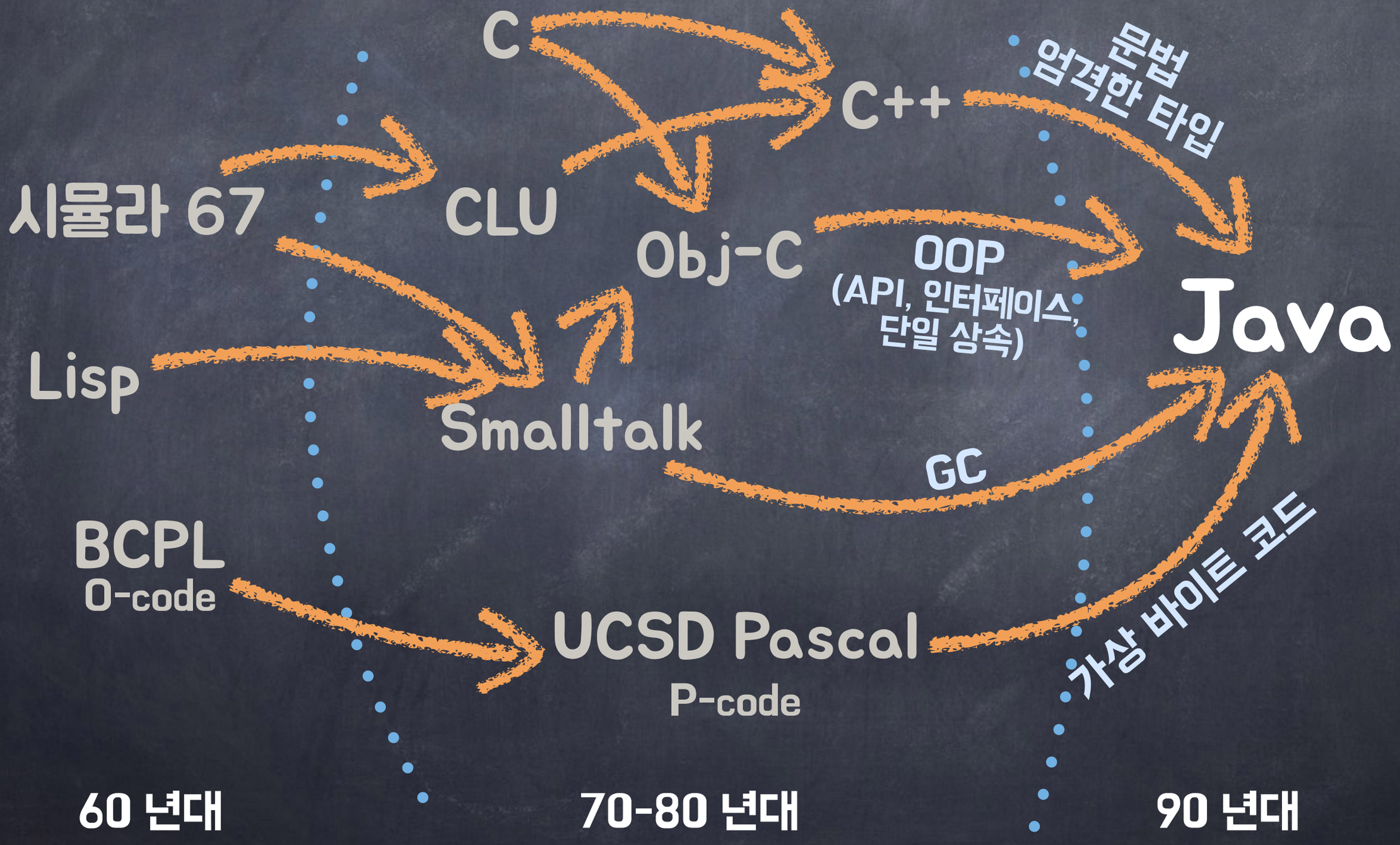
2. 객체지향 프로그래밍

3. 견고하고 안전한 = 엄격한 정적 타입

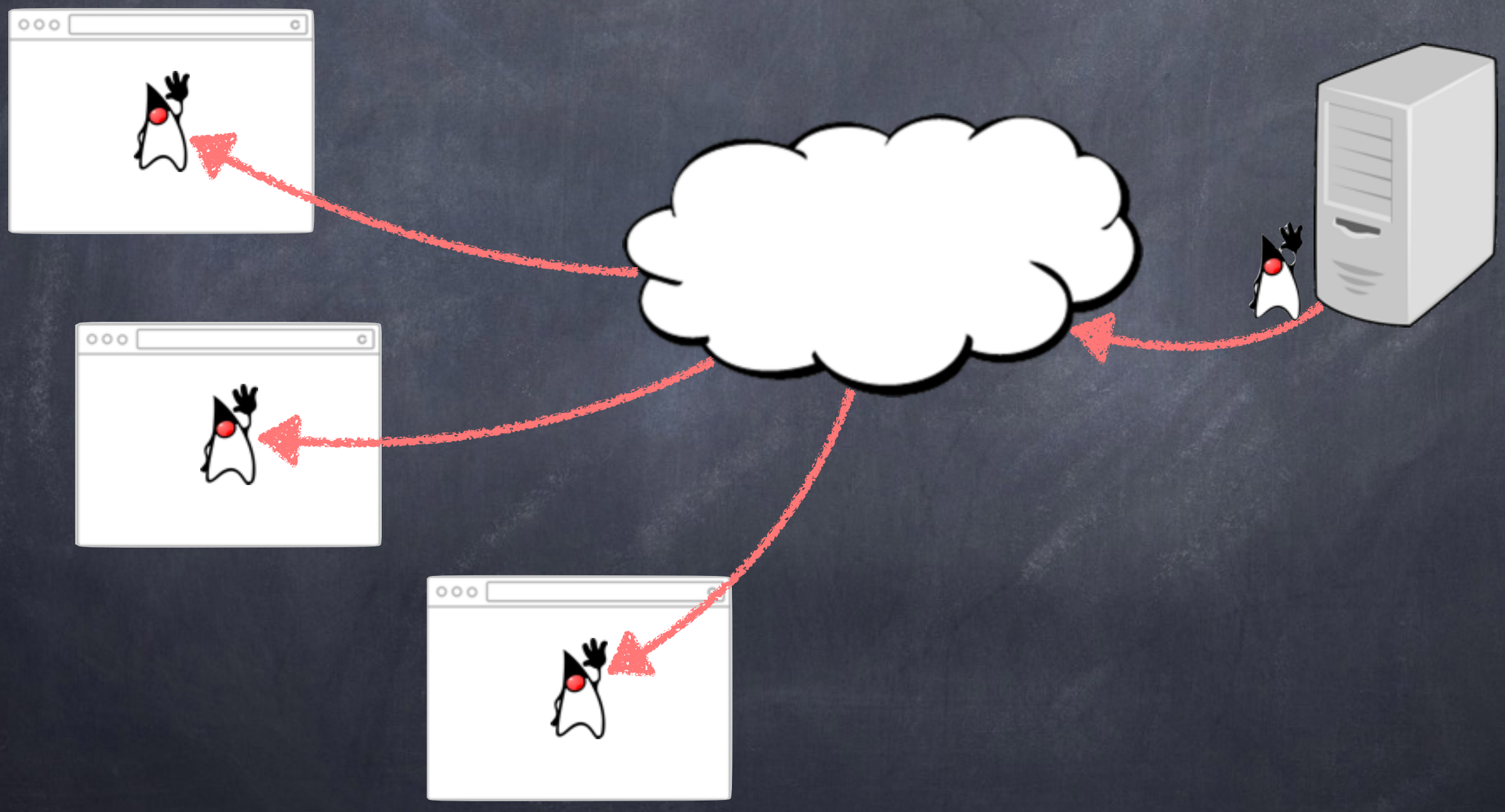
4. 인터프리터, 아키텍처 중립성, 이식성, 동적 로딩

5. 고성능, 멀티쓰레드

자바의 계보


C
C++
문법
엄격한 타입
시뮬라 67
CLU
Obj-C
OOP
(API, 인터페이스,
단일 상속)
Java
Lisp
Smalltalk
GC
BCPL
O-code
UCSD Pascal
P-code
가상 바이트 코드
60 년대
70-80 년대
90 년대

인터넷, 앱 배포 경로



자바 1.2, 1.3

기업용 플랫폼 강화

J2SE 1.2
- Java 2,
- J2SE, J2EE, J2ME
- Java Plugins
- 컬랙션 프레임워크
- 스윙
- 고성능, 멀티쓰레드

1998 JPE 발표(EJB1.0)
1999 J2EE 1.2 발표
- JDBC 2.0
- JNI 1.2
- Servlet 2.2
- JSP 1.1
- EJB 1.1
- JMS 1.0
- JTA 1.0
- JavaMail 1.1
- JAF 1.0

J2SE 1.3
- 핫 자바 VM 기본 적용 -> 중요 기업용 시장 진출

자바 1.4, 5, 6

언어 현대화, .Net 경쟁력 강화

J2SE 1.4

1. Java Plugin
2. New I/O
3. 정규식
4. JAXP

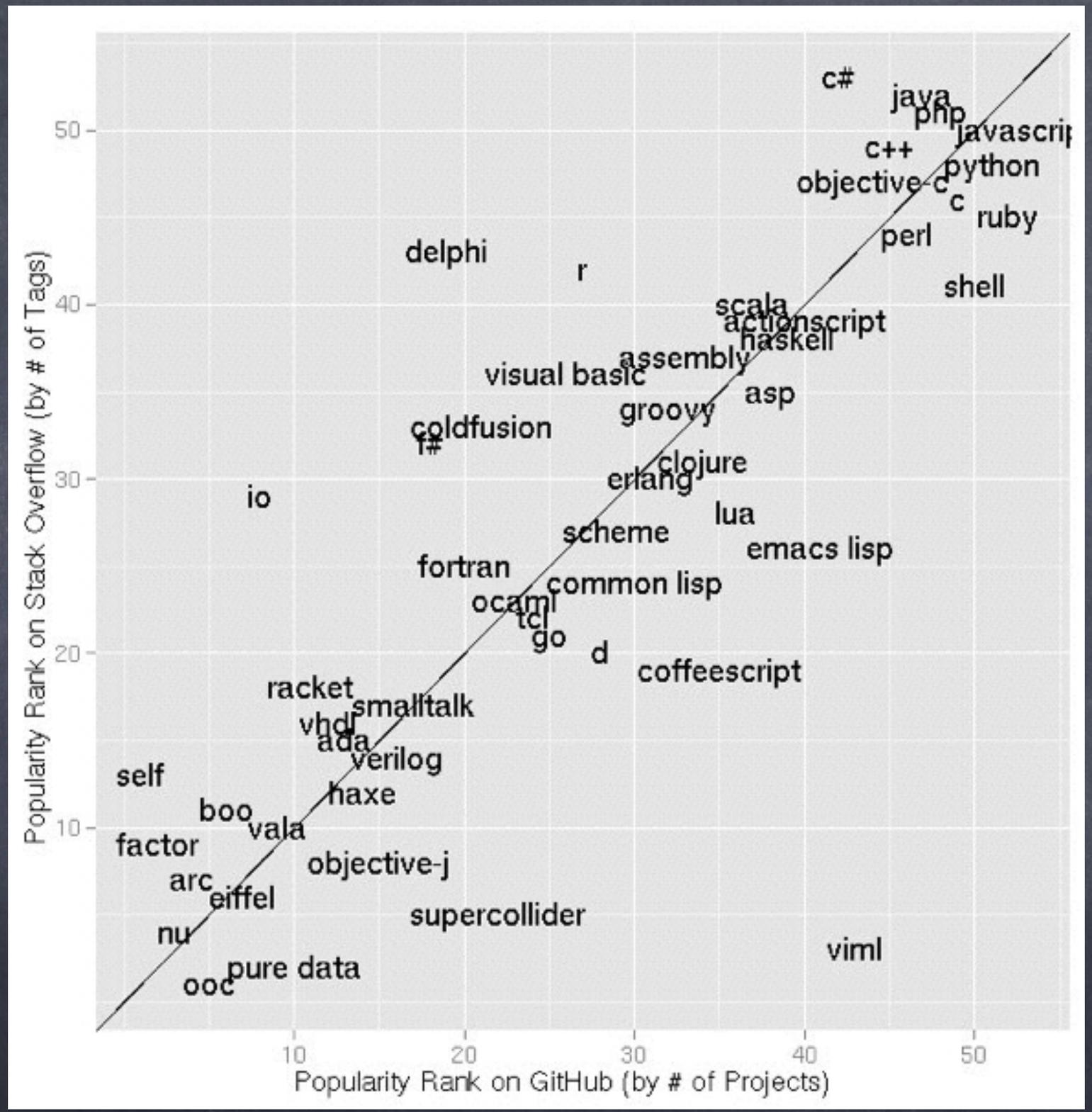
Java SE 6

1. 스크립트 지원
2. 컴파일러 API
3. 성능 개선

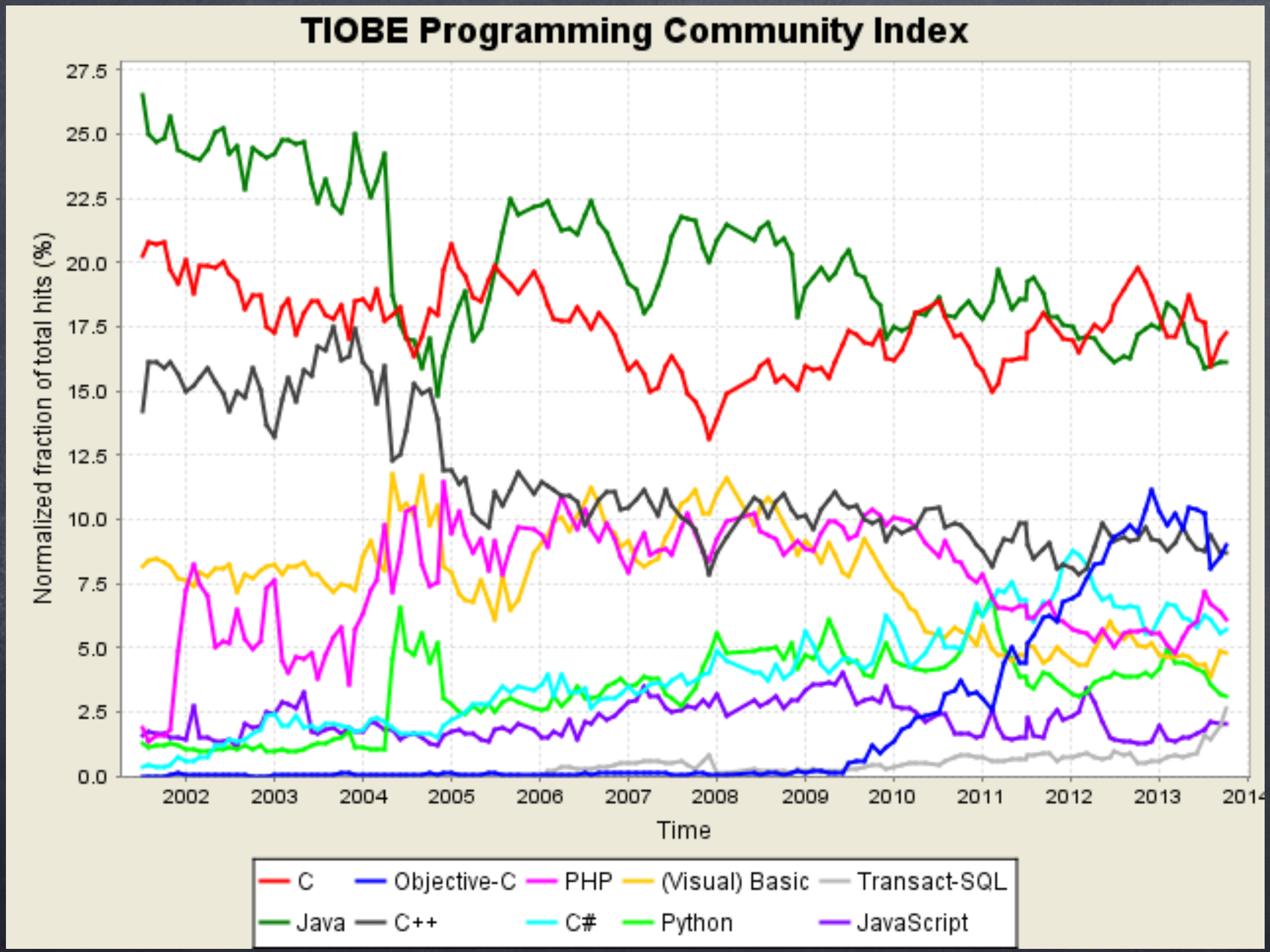
J2SE 5.0

1. 지네릭
2. 어노테이션
3. 오토 박싱
4. 열거식
5. 가변 변수
6. for each 반복문
7. 자바 메모리 모델 수정
8. 정적 임포트

Position Mar 2010	Position Mar 2009	Delta in Position	Programming Language	Ratings Mar 2010	Delta Mar 2009	Status
1	1	=	Java	17.509%	-2.29%	A
3	4	↑	PHP	9.908%	+0.42%	A
4	3	↓	C++	9.610%	-0.75%	A
5	5	=	(Visual) Basic	6.574%	-1.71%	A
6	7	↑	C#	4.264%	-0.06%	A
7	6	↓	Python	4.230%	-0.95%	A
8	9	↑	Perl	3.821%	+0.40%	A
9	10	↑	Delphi	2.684%	-0.03%	A
10	8	↓↓	JavaScript	2.651%	-0.96%	A
11	11	=	Ruby	2.327%	-0.27%	A
12	32	↑↑↑↑↑↑↑↑↑↑	Objective-C	1.970%	+1.79%	A
13	-	↑↑↑↑↑↑↑↑↑↑	Go	0.921%	+0.92%	A
14	15	↑	SAS	0.769%	-0.03%	A
15	13	↓↓	PL/SQL	0.737%	-0.31%	A
16	22	↑↑↑↑↑↑	MATLAB	0.661%	+0.20%	B
17	17	=	ABAP	0.639%	+0.00%	B
18	16	↓↓	Pascal	0.603%	-0.13%	B
19	19	=	ActionScript	0.594%	+0.11%	B
20	27	↑↑↑↑↑↑↑	Fortran	0.563%	+0.24%	B


c#
java
php
javascrip
c++
python
objective-c
c
ruby
perl
delphi
r
shell
scala
actionscript
haskell
assembly
visual basic
asp
groovy
coldfusion
f#
clojure
erlang
io
lua
scheme
emacs lisp
fortran
common lisp
ocaml
tcl
go
d
coffeescript
racket
smalltalk
vhdl
ada
verilog
self
haxe
boo
vala
factor
objective-j
arc
eiffel
supercollider
nu
viml
pure data
ooc
10
20
30
40
50
Popularity Rank on GitHub (by # of Projects)
Popularity Rank on Stack Overflow (by # of Tags)

침체기?


TIOBE Programming Community Index
Normalized fraction of total hits (%)
27.5
25.0
22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
Time
C
Objective-C
PHP
(Visual) Basic
Transact-SQL
Java
C++
C#
Python
JavaScript

자바 혁신 생태계

오픈 소스의 세대 발전

3세대
메타-조직
(meta-organization)
Eclipse, OpenJDK

2세대
대규모 공동체
Aache Software Foundation

1세대
느슨하게 연결된 개인
GNU, Linux

8 — OpenJDK JDK 8 출시

JCP.Next 제안 — 더그 리 교수, Open JDK 운영 위원 당선

7 — JUG JCP 합류 — OpenJDK, IBM합류, JDK 7출시

오라클, 썬 인수 — 자바 100% 오픈소스화, ASF JCP 탈퇴

ASF voted No!

레드햇 100% 오픈소스 자바 공개 TDK 통과

자바 라이브러리 오픈소스화

6 — 자바 오픈소스화 발표

Jakarta 해체 시작

5 — JBoss 4.0(최초 EJB 3), 스프링 1.0 출시

JBoss JCP 합류 — Jboss 인증 획득, 스프링 공개

1.4 — JCP, JDK 개선 — MS .Net Framework 출시

1.3 — JCP 2.0 개정 — ASF, JCP 합류

Jakarta 시작

1.2 — JCP 설립

1.1

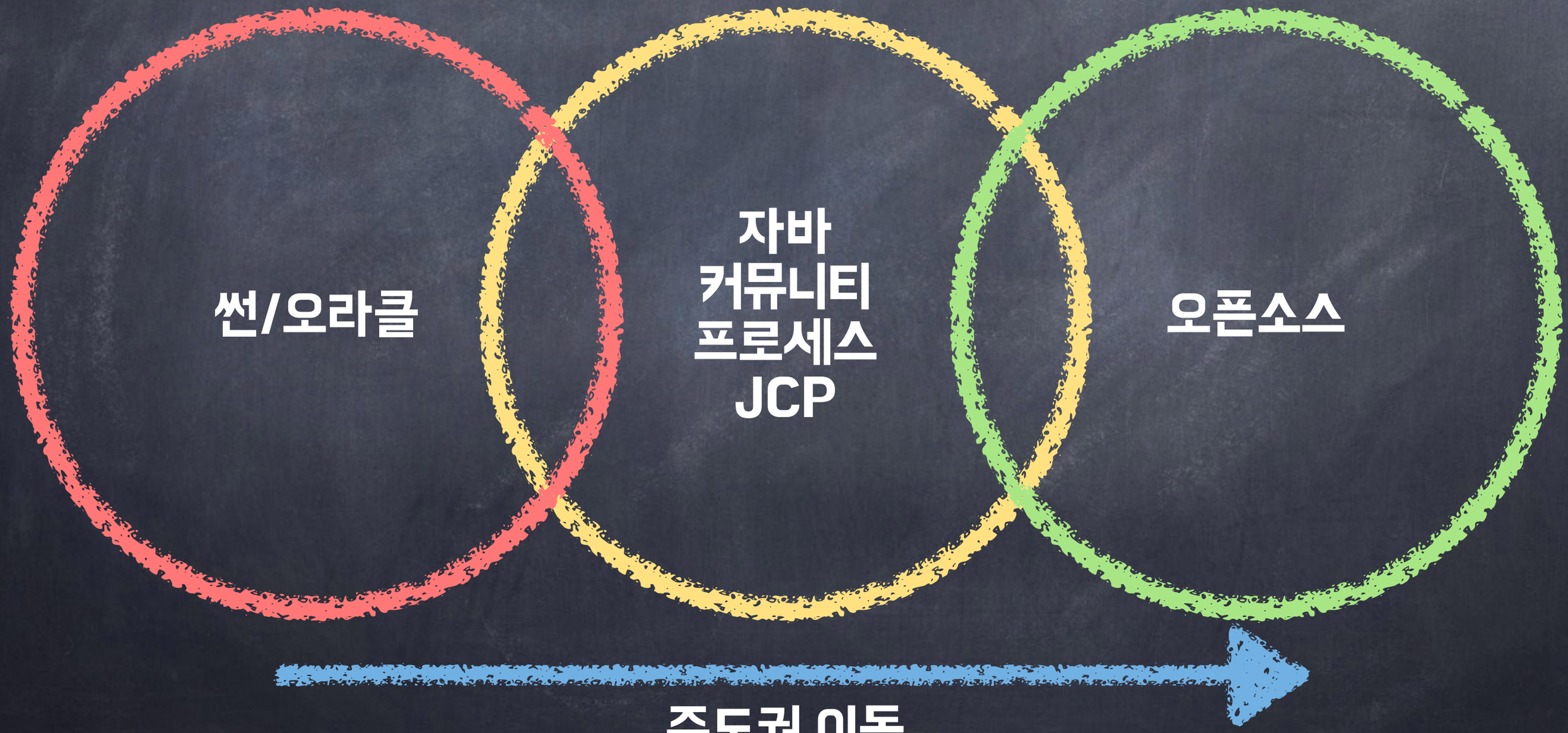
1.0

β

JCP
이해관계자
중심
개방형 혁신

오픈소스
개방형 혁신

자바 혁신 생태계



- Oracle, SUN 인수
- JDK 오픈소스화
- OpenJDK 중심 JDK 개발
- 람다식

β 1.0 1.1 1.2 1.3 1.4 5 6 7 8

나도 모르게

찜찜

β 1.0 1.1 1.2 1.3 1.4 5 6 7 8

자바의 현실은 이것으로 끝인가?

발할라(Valhalla)

JVM 개선 프로젝트

- 값 객체(Value Object; JEP 169)
  - 기본 타입(int, boolean, byte 등)과 같이 값으로 취급되는 객체
  - 사용자 정의 가능한, 참조 없는 새로운 타입
  - “Codes like a class, works like an int!”
- 지네릭 특화(Generic Specialization)
  - 기본 타입(int, boolean, byte 등)을 지네릭에 사용: List<int>
  - 오토박싱 대신 값 객체 사용, 성능 향상
- 구체화된 지네릭
  - 오토박싱 대신 값 객체 사용
- 개량된 ‘volatile’(JEP 193)
- 클래스 다이나믹

기타

- 파나마(Panama): 차세대 JNI 개발 프로젝트
  http://openjdk.java.net/projects/panama/
- 어노테이션 파이프라인 2.0:
  javac 컴파일 파이프라인의 어노테이션 처리 기능 개선
  http://openjdk.java.net/projects/anno-pipeline/
- Javadoc.next
  Doclet API 단순화(JEP 221), HTML5 생성(JEP 224),
  Javadoc 내 기본 검색
- Device I/O: 임베디드 기기의 GPIO, I2C, UART, SPI 접근 API
- Jigsaw: JDK 모듈화, 성능/보안성 향상
- Penrose: Jigsaw와 OSGi의 상호연동

Beyond Java

Beyond Old Java

어떤 상황에서도
'승리'를 찾는다

https://www.facebook.com/groups/springkorea/
https://groups.google.com/forum/#!forum/ksug