

# Basic of Java





Somkiat Puisungnoen

Search

Somkiat | Home

Update Info 1 View Activity Log 10+ ...

Timeline About Friends 3,138 Photos More

When did you work at Opendream? X

... 22 Pending Items

Post Photo/Video Live Video Life Event

What's on your mind?

Public Post

Intro

Software Craftsmanship

Software Practitioner at สยามชานนาญกิจ พ.ศ. 2556

Agile Practitioner and Technical at SPRINT3r

Somkiat Puisungnoen 15 mins · Bangkok · ...

Java and Bigdata

 Basic of Java

© 2017 - 2018 Siam Chamnankit Company Limited. All rights reserved.

3

somkiat.cc

Page Messages Notifications 3 Insights Publishing Tools Settings Help ▾

somkiat.cc  
@somkiat.cc

Home Posts Videos Photos

Liked Following Share ...

+ Add a Button



# Agenda Day 1

- All about Java Programmer mistakes
- Object-Oriented Design
- SOLID
- Workshop



# Mistake of Java Programmer



# **Set JAVA\_HOME**



# How to build and compile ?



# Compile and Run

```
$javac -version  
$java -version
```



# How to read the Java document

## ?

<https://docs.oracle.com/javase/8/docs/>



# Build tools

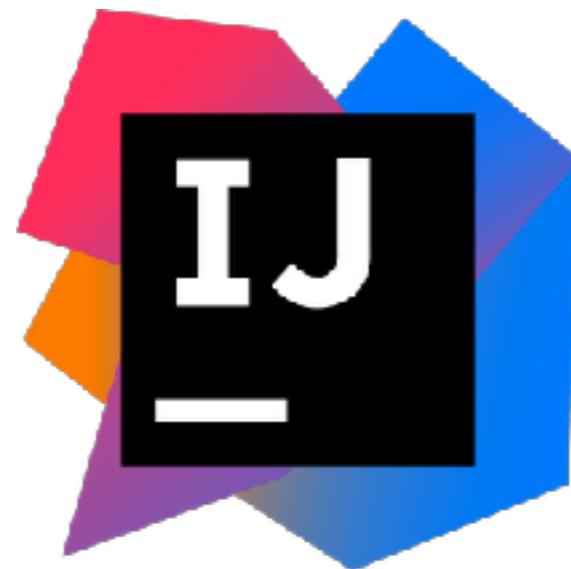


**Maven™**

 gradle

The Gradle logo consists of a green hexagonal icon with a white circle in the center, followed by the word "gradle" in a large, lowercase, green sans-serif font.

# IDE WARs



# **Trust in your code ?**



# Your code not break anything ?



# **Write Unit tests**

**JUnit**



# Hello with Unit testing



# **Basic of Java**



# Basic of Java

Working with String  
if-else statement

Looping => for/for each, while and do-while



# Basic of Java

if-else statement

Looping => for/for each, while and do-while



# Workshop online



# CyberDojo

 **cyber-dojo**

**the place to practise programming**

**i'm on my own**

**we're in a group**

commercial use of [cyber-dojo.org](http://cyber-dojo.org) requires a licence

non-commercial use is free but

**please donate**

<http://www.cyber-dojo.org/>



# Day 2



# **Run your code/test in command line**



# **Check before run !!**

**For Windows OS**

```
$echo %JAVA_HOME%
```

**For Mac/Linux**

```
$echo $JAVA_HOME
```



# Run test with maven

\$mvnw clean test



# Build JAR file with maven

\$mvnw clean package



# Try to compile and win without Apache Maven



# Compile

```
$javac -s src/main/java -d your_target  
src/main/java/com/example/demo/HelloDay2.java
```

*-s = folder of source code*

*-d = folder of class files (destination)*



# Run

```
$java -cp your_target  
com.example.demo.HelloDay2
```

*-cp = classpath of class/JAR files*



# Code Smell & Refactoring

<https://sourcemaking.com/refactoring/smells>



# Code Smell



# Code Smell



*“Bad Code Smells are symptoms of poor design or implementation choices”*

[Martin Fowler]



# Code Smell Workshop

<https://github.com/emilybache/Tennis-Refactoring-Kata>



# Composition over Inheritance

[https://en.wikipedia.org/wiki/Composition\\_over\\_inheritance](https://en.wikipedia.org/wiki/Composition_over_inheritance)



# Composition over Inheritance

HAS-A

IS-A



# Source code management



# Download and Install Git

The screenshot shows the official Git website at <https://git-scm.com/>. At the top left is the Git logo with the tagline "distributed-is-the-new-centralized". A search bar is at the top right. The main content area features two columns of links: "About", "Documentation", "Downloads", and "Community". To the right, there's a diagram illustrating the distributed nature of Git with multiple repositories connected by red lines, and a large monitor displaying the latest source release information.

**git** --distributed-is-the-new-centralized

Search entire site...

Git is a **free and open source** distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is **easy to learn** and has a **tiny footprint with lightning fast performance**. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like **cheap local branching**, convenient **staging areas**, and **multiple workflows**.

 Learn Git in your browser for free with [Try Git](#).

**About**  
The advantages of Git compared to other source control systems.

**Documentation**  
Command reference pages, Pro Git book content, videos and other material.

**Downloads**  
GUI clients and binary releases for all major platforms.

**Community**  
Get involved! Bug reporting, mailing list, chat, development and more.

Latest source Release  
**2.18.0**  
Release Notes (2018-06-21)  
[Download 2.17.1 for Mac](#)

<https://git-scm.com/>



# Git command line

\$git status

\$git init

\$git add <file>

\$git commit -m “Reason to commit”

\$git push <remote> <local branch>

\$git pull <remote> <local branch>

\$git remote add <name> <url>



# Create repository at Github

## Create a new repository

A repository contains all the files for your project, including the revision history.

Owner      Repository name

 up1  

Great repository names are short and memorable. Need inspiration? How about [bookish-octo-tribble](#).

---

Description (optional)

---

 **Public**  
Anyone can see this repository. You choose who can commit.

 **Private**  
You choose who can see and commit to this repository.

---

**Initialize this repository with a README**  
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

---

Add .gitignore: **None** ▾ | Add a license: **None** ▾ 

---

**Create repository**



# Create already

The screenshot shows a GitHub repository page for 'up1 / sample'. The top navigation bar includes links for Code, Issues (0), Pull requests (0), Projects (0), Wiki, Insights, and Settings. A 'Unwatch' button with a count of 1 is also present. The main content area features a 'Quick setup — if you've done this kind of thing before' section with options to 'Set up in Desktop' or choose between HTTPS and SSH, providing the URL <https://github.com/up1/sample.git>. Below this, a note says 'We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#)'. Further down, instructions for creating a new repository on the command line are shown in a red-bordered box:

```
echo "# sample" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/up1/sample.git
git push -u origin master
```



# Working with Git



# 1. Create repos on your folder

```
$git init
```



## 2. Add file/folder to git repos

\$git add <filename>

\$git add <folder>



# 3. Commit your change

```
$git commit -m "<message>"
```



# 4. Add remote repository

\$git remote add <name> <remote url>

\$git remote add **origin** **https://github.com/up1/sample.git**



# 5. Push your change to remote repos

```
$git push origin master
```

And check your code at [github.com](https://github.com)



# Day 3



# OOP with Harry Potter



# Design

Book

BookItem

Order

PriceCalculator

DiscountCalculator



# Properties/Behaviors

Book

BookItem

Order

PriceCalculator

DiscountCalculator



# Relationships ?

Book

BookItem

Order

PriceCalculator

DiscountCalculator



# Develop



# Develop

1. Create book class
2. Create Order class
3. Create BookItem class
4. Create Price Calculator
5. Create Discount Calculator



# 1. Create Book



# How to create instance ?

1. With constructor
2. With Creation Method
3. With Builder pattern



# Create Book

```
@Test
```

```
public void createBookWithConstructor() {  
    Book book1 = new Book("H1", 8, 2);  
}
```

```
@Test
```

```
public void createBookWithCreationMethod() {  
    Book book1 = Book.createHarryPotterOne();  
    Book book2 = Book.createHarryPotterTwo();  
}
```

```
@Test
```

```
public void createBookWithBuilderPattern() {  
    Book book1 = new BookBuilder()  
        .setPrice(8)  
        .setName("H1")  
        .build();
```



## 2. Create Order



# Create Order

One order can have 0-N item(s)

Order

BookItem 1

Book 1

BookItem 2

Book 2

BookItem 3

Book 3



# Start with Empty Order

```
@Test  
public void createEmptyOrder() {  
    Order order = new Order();  
    order.process();  
    assertEquals(0, order.getBookAmount());  
    assertEquals(0, order.getTotalPrice(), 0.00);  
    assertEquals(0, order.getDiscount(), 0.00);  
    assertEquals(0, order.getNetPrice(), 0.00);  
}
```



# Order with 1 item

```
@Test  
public void createOrderWithOneBook() {  
    Book book1 = new Book("H1", 8, 2);  
    BookItem bookItem = new BookItem(book1, 1);  
    Order order = new Order();  
    order.addItem(bookItem);  
    order.process();  
  
    assertEquals(1, order.getBookAmount());  
    assertEquals(8, order.getTotalPrice(), 0.00);  
    assertEquals(0, order.getDiscount(), 0.00);  
    assertEquals(8, order.getNetPrice(), 0.00);  
}
```



# Order with 2 item = 5%

```
@Test  
public void createOrderWithTwoBook() {  
    Book book1 = new Book("H1", 8, 2);  
    Book book2 = new Book("H2", 8, 2);  
    BookItem bookItem1 = new BookItem(book1, 1);  
    BookItem bookItem2 = new BookItem(book2, 1);  
    Order order = new Order();  
    order.addItem(bookItem1);  
    order.addItem(bookItem2);  
    order.process();  
  
    assertEquals(2, order.getBookAmount());  
    assertEquals(16, order.getTotalPrice(), 0.00);  
    assertEquals("Discount not correct", 16 * 0.05, order.getDiscount(), 0.00);  
    assertEquals(16 - (16 * 0.05), order.getNetPrice(), 0.00);  
}
```



# 3. Create BookItem



# 4. Price Calculation



# 5. Discount Calculation

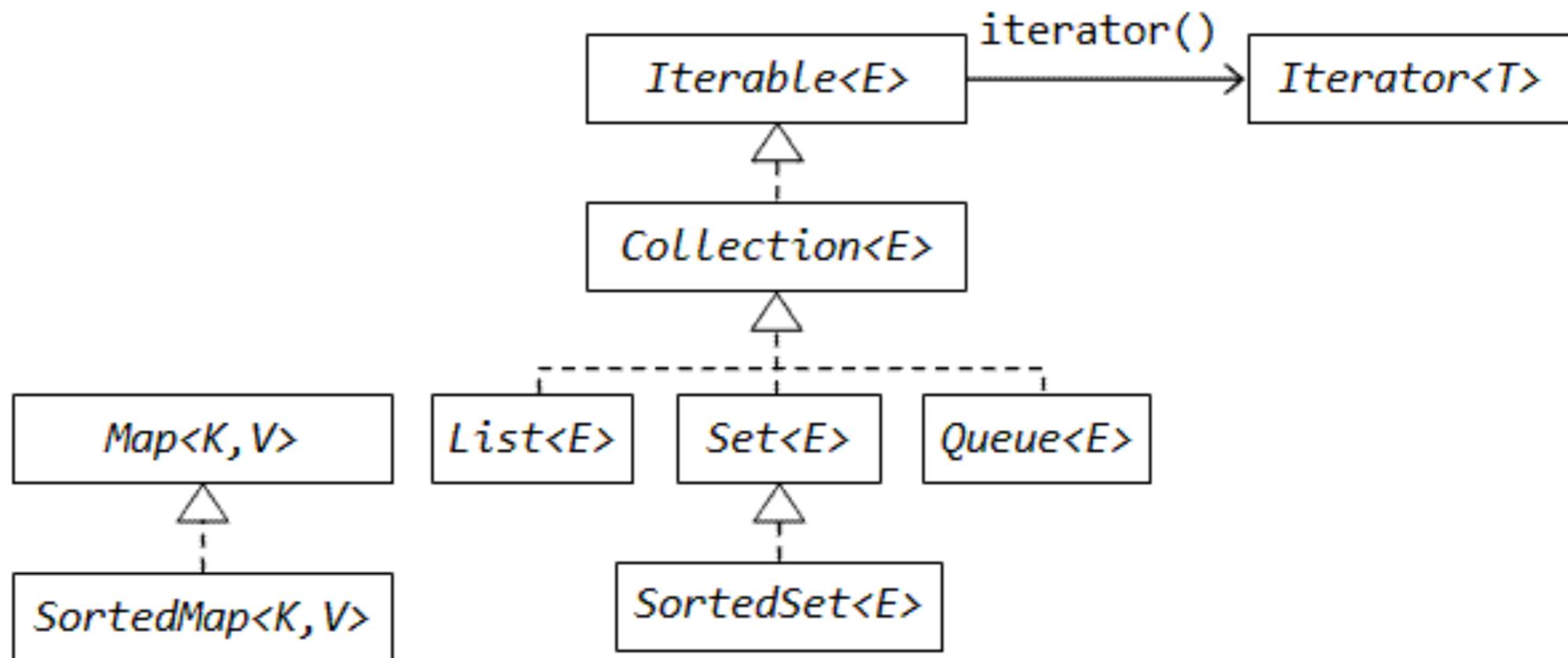


# Java Collection Framework

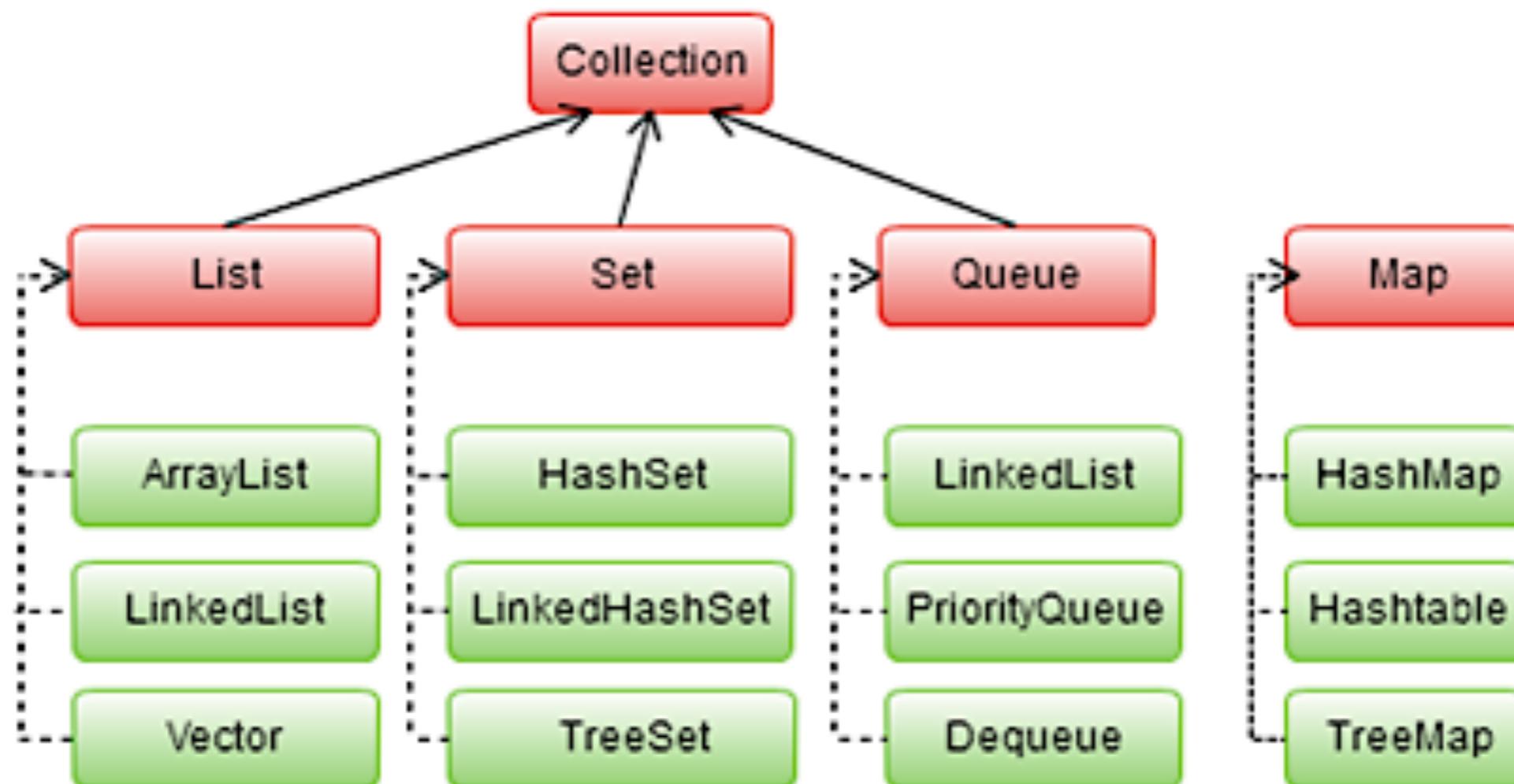
<https://docs.oracle.com/javase/8/docs/technotes/guides/collections/overview.html>



# Java Collection Framework



# Java Collection Framework



# Java Collections Cheat Sheet



## Notable Java collections libraries

### Fastutil

<http://fastutil.di.unimi.it/>

Fast & compact type-specific collections for Java  
Great default choice for collections of primitive types, like int or long. Also handles big collections with more than  $2^{31}$  elements well.

### Guava

<https://github.com/google/guava>

Google Core Libraries for Java 6+  
Perhaps the default collection library for Java projects. Contains a magnitude of convenient methods for creating collection, like fluent builders, as well as advanced collection types.

### Eclipse Collections

<https://www.eclipse.org/collections/>

Features you want with the collections you need  
Previously known as gs-collections, this library includes almost any collection you might need: primitive type collections, multimap, bidirectional maps and so on.

### JCTools

<https://github.com/JCTools/JCTools>

Java Concurrency Tools for the JVM.  
If you work on high throughput concurrent applications and need a way to increase your performance, check out JCTools.

## What can your collection do for you?

Collection class	Thread-safe alternative	Your data				Operations on your collections						
		Individual elements	Key-value pairs	Duplicate element support	Primitive support	FIFO	Sorted	LIFO	Perform 'contains' check	By key	By value	By index
HashMap	ConcurrentHashMap	✗	✓	✗	✗	✗	✗	✗	✓	✓	✗	✗
HashBiMap (Guava)	Maps.synchronizedBiMap(new HashBiMap())	✗	✓	✗	✗	✗	✗	✗	✓	✓	✓	✗
ArrayListMultimap (Guava)	Maps.synchronizedMultiMap(new ArrayListMultimap())	✗	✓	✓	✗	✗	✗	✗	✓	✓	✗	✗
LinkedHashMap	Collections.synchronizedMap(new LinkedHashMap())	✗	✓	✗	✗	✓	✗	✗	✓	✓	✗	✗
TreeMap	ConcurrentSkipListMap	✗	✓	✗	✗	✗	✓	✗	✓*	✓*	✗	✗
Int2IntMap (Fastutil)		✗	✓	✗	✓	✗	✗	✗	✓	✓	✗	✓
ArrayList	CopyOnWriteArrayList	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗	✓
HashSet	Collections.newSetFromMap(new ConcurrentHashMap<>())	✓	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗
IntArrayList (Fastutil)		✓	✗	✓	✓	✓	✓	✗	✓	✗	✗	✓
PriorityQueue	PriorityBlockingQueue	✓	✗	✓	✗	✗	✓**	✗	✗	✗	✗	✗
ArrayDeque	ArrayBlockingQueue	✓	✗	✓	✗	✓**	✗	✓**	✗	✗	✗	✗

\*  $O(\log n)$  complexity, while all others are  $O(1)$  - constant time

\*\* when using Queue interface methods: offer() / poll()

## How fast are your collections?

Collection class	Random access by index / key	Search / Contains	Insert
ArrayList	$O(1)$	$O(n)$	$O(n)$
HashSet	$O(1)$	$O(1)$	$O(1)$
HashMap	$O(1)$	$O(1)$	$O(1)$
TreeMap	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$

Remember, not all operations are equally fast. Here's a reminder of how to treat the Big-O complexity notation:

$O(1)$  - constant time, really fast, doesn't depend on the size of your collection

$O(\log(n))$  - pretty fast, your collection size has to be extreme to notice a performance impact

$O(n)$  - linear to your collection size: the larger your collection is, the slower your operations will be

BROUGHT TO YOU BY  
**JRebel**

[http://files.zeroturnaround.com/pdf/zt\\_java\\_collections\\_cheat\\_sheet.pdf](http://files.zeroturnaround.com/pdf/zt_java_collections_cheat_sheet.pdf)



# Types of Error in Java

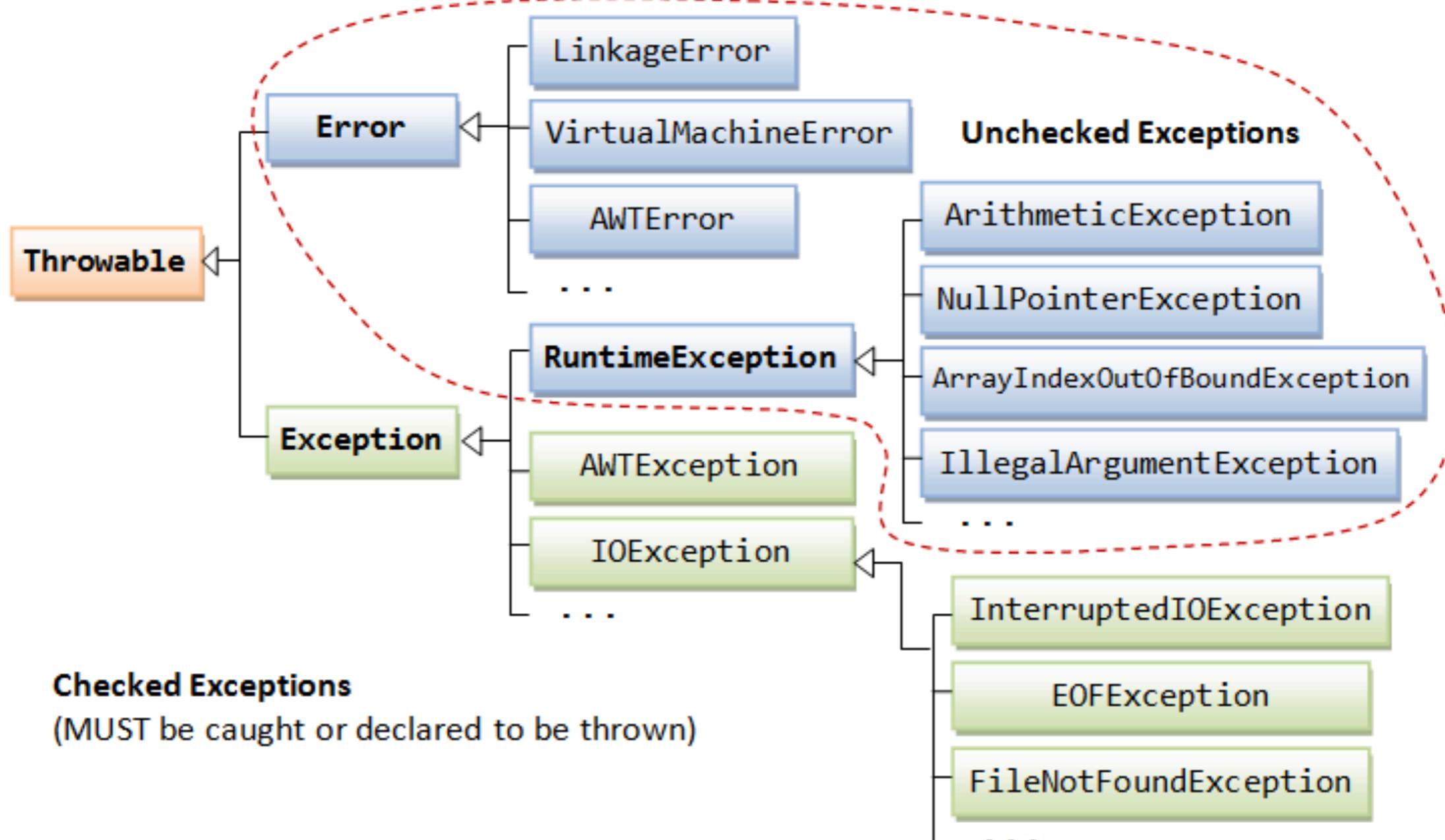


# Types of Errors

- Compile-time errors
- Runtime errors
- Logical errors



# Exception classes



# Day 4



# Features in Java 8



# Java 5 (Tiger) in 2005

Largest change of Java

Generic

Enhance for-loop

Annotation

Type-safe enumeration

Concurrency enhancement (AtomicInteger)



# **10 years later ...**



# **Java 8 is Massive changes !!**



# Java 8 Best Practices Cheat Sheet



BROUGHT TO YOU BY



## Default methods

Evolve interfaces & create traits

```
// Default methods in interfaces
@FunctionalInterface
interface Utilities {
    default Consumer<Runnable> m() {
        return (r) -> r.run();
    }

    // default methods, still functional
    Object function(Object o);
}

class A implements Utilities { // implement
    public Object function(Object o) {
        return new Object();
    }

    // call a default method
    Consumer<Runnable> n = new A().m();
}
}
```

Traits: 1 default method per interface  
Don't enhance functional interfaces  
Only conservative implementations

## Lambdas

Syntax:

```
(parameters) -> expression
(parameters) -> { statements; }
```

```
// takes a Long, returns a String
Function<Long, String> f = (l) -> l.toString();

// takes nothing gives you Threads
Supplier<Thread> s = Thread::currentThread;

// takes a string as the parameter
Consumer<String> c = System.out::println;

// use them with streams
new ArrayList<String>().stream();

// peek: debug streams without changes
peek(e -> System.out.println(e));

// map: convert every element into something
map(e -> e.hashCode());

// filter: pass some elements through
filter(e -> ((e.hashCode() % 2) == 0));

// collect all values from the stream
collect(Collectors.toCollection(TreeSet::new))
```

## java.util. Optional

A container for possible null values

```
// Create an optional
Optional<String> optional =
Optional.ofNullable(a);

// process the optional
optional.map(s -> "RebellLabs:" + s);

// map a function that returns Optional
optional.flatMap(s -> Optional.ofNullable(s));

// run if the value is there
optional.ifPresent(System.out::println);

// get the value or throw an exception
optional.get();

// return the value or the given value
optional.orElse("Hello world!");

// return empty Optional if not satisfied
optional.filter(s -> s.startsWith("RebellLabs"));
```

## Rules of Thumb

Expressions over statements  
Refactor to use method references  
Chain lambdas rather than growing them

Fields - use plain objects  
Method parameters, use plain objects  
Return values - use Optional  
Use `orElse()` instead of `get()`

[http://files.zeroturnaround.com/pdf/zt\\_java8\\_best\\_practices.pdf](http://files.zeroturnaround.com/pdf/zt_java8_best_practices.pdf)



# Lambda expression



# Lambda expression

Functionality as a method arguments

Code as data

Help code more compact



# Demo with Sorting data

Sorting data in List with Comparator

Employee id=1 name=First

Employee id=2 name=Second

Employee id=3 name=Third



# Create class Employee

```
class Employee {  
    private int id;  
    private String name;  
    private double salary;  
  
    //Setter and Getter methods
```



# Sorting with java.util.Comparator

```
Comparator<Employee> sortByName = new Comparator<Employee>() {  
    @Override  
    public int compare(Employee e1, Employee e2) {  
        return e1.getName().compareTo(e2.getName());  
    }  
};
```



# Using with Collections.sort()

```
List<Employee> list = new ArrayList<>();  
list.add(new Employee(500, "First", 150000));  
list.add(new Employee(504, "Second", 120000));  
list.add(new Employee(503, "Third", 100000));  
list.add(new Employee(730, "Forth", 45000));  
  
Collections.sort(list, sortByName);  
list.forEach(System.out::println);
```



# Run and see output

```
demo.java8.lambda.Employee@e9e54c2  
demo.java8.lambda.Employee@65ab7765  
demo.java8.lambda.Employee@1b28cdfa  
demo.java8.lambda.Employee@eed1f14
```

*Try to improve format of output !!  
more readable*



# Override `toString()`

```
class Employee {  
    private int id;  
    private String name;  
    private double salary;  
  
    @Override  
    public String toString() {  
        return "Employee [id=" + id + ", name=" + name + ", salary=" +  
salary + "]";  
    }  
}
```

```
Employee [id=500, name=First, salary=150000.0]  
Employee [id=504, name=Second, salary=120000.0]  
Employee [id=503, name=Third, salary=100000.0]  
Employee [id=730, name=Forth, salary=45000.0]
```



# Sorting with Lambda expression

```
Comparator<Employee> sortByNameWithLambda =  
    (Employee e1, Employee e2) ->  
        e1.getName().compareTo(e2.getName());  
  
Collections.sort(list, sortByName);  
list.forEach(System.out::println);
```



# Workshop

Sort by name (Z to A)

Sort by ID (0 to 9)



# Java Stream API



# Java Stream API

A new package **java.util.stream**  
New functionality which contains classes for  
processing sequences of elements



# Java Stream API

## Working with List and Array

```
String[] datas = new String[] {"P", "U", "I"};
Stream<String> streams = Arrays.stream(datas);
streams.forEach(System.out::println);
```

```
Stream<String> streams2 = Stream.of("P", "U", "I");
streams2.forEach(System.out::println);
```



# Java Stream API

## Sequential and Parallel

```
List<String> myList = Arrays.asList("P", "U", "I");
```

```
myList.stream().forEach(System.out::println);
```

```
myList.parallelStream().forEach(System.out::println);
```



# Operations

Iterating  
Filtering  
Mapping  
Matching  
Collecting



# Iterating

```
myList.stream().forEach(System.out::println);
```



# Filtering

```
myList.stream()  
    .filter(element -> element.contains("P"))  
    .forEach(System.out::println);
```



# Matching

Try to validate all elements of sequence

```
boolean isValid = myList.stream()  
    .anyMatch(element -> element.contains("P"));  
  
out.println(isValid);
```



# Mapping

Try to convert elements by apply a special function

```
List<String> results =  
    myList.stream().map(element -> element.toLowerCase())  
        .collect(Collectors.toList());  
  
out.println(results);
```



# Collecting

Try to convert stream to collection or map

```
List<String> results =  
    myList.stream().map(element -> element.toLowerCase())  
        .collect(Collectors.toList());  
  
out.println(results);
```



# Workshop

United States of America => **USA**

united states of america => **USA**

Light Amplification by Stimulation of Emitted Radiation => **LASER**

Jordan Of the World => **JTW**



# SOLID principle



# SOLID principle



**SOLID**

Software development is not a Jenga game.



# 1. Single Responsibility Principle

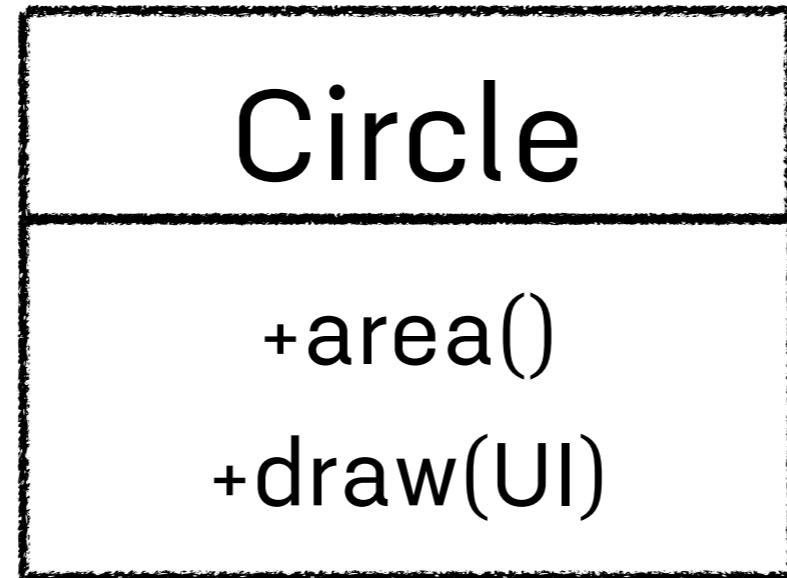


## Single Responsibility Principle

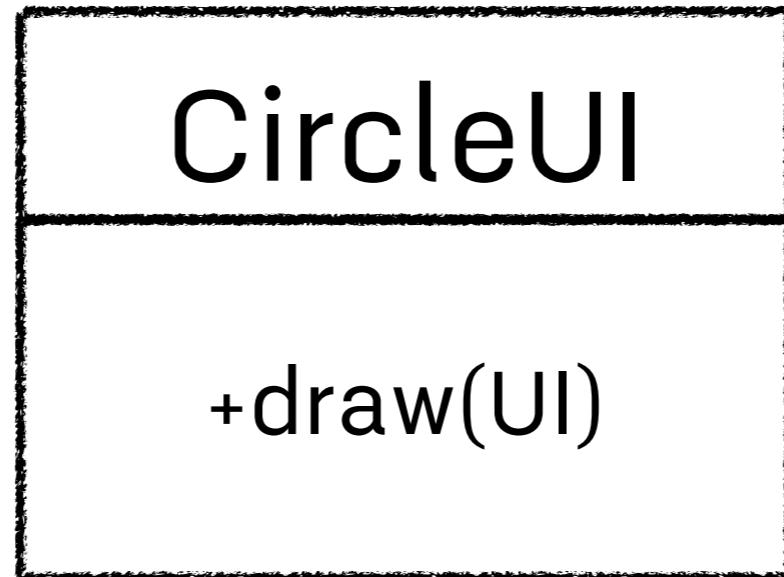
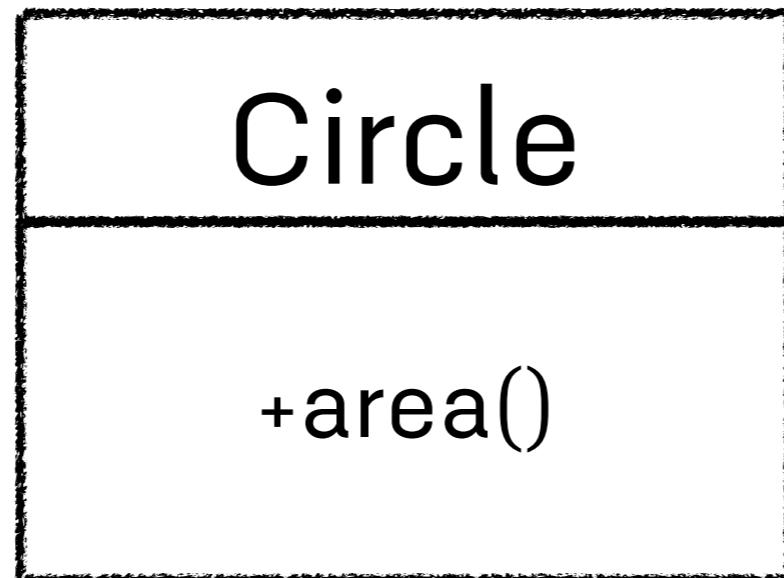
Just because you *can* doesn't mean you *should*.



# Example



# Example



## 2. Open-Closed Principle



# Example

DrawingEditor

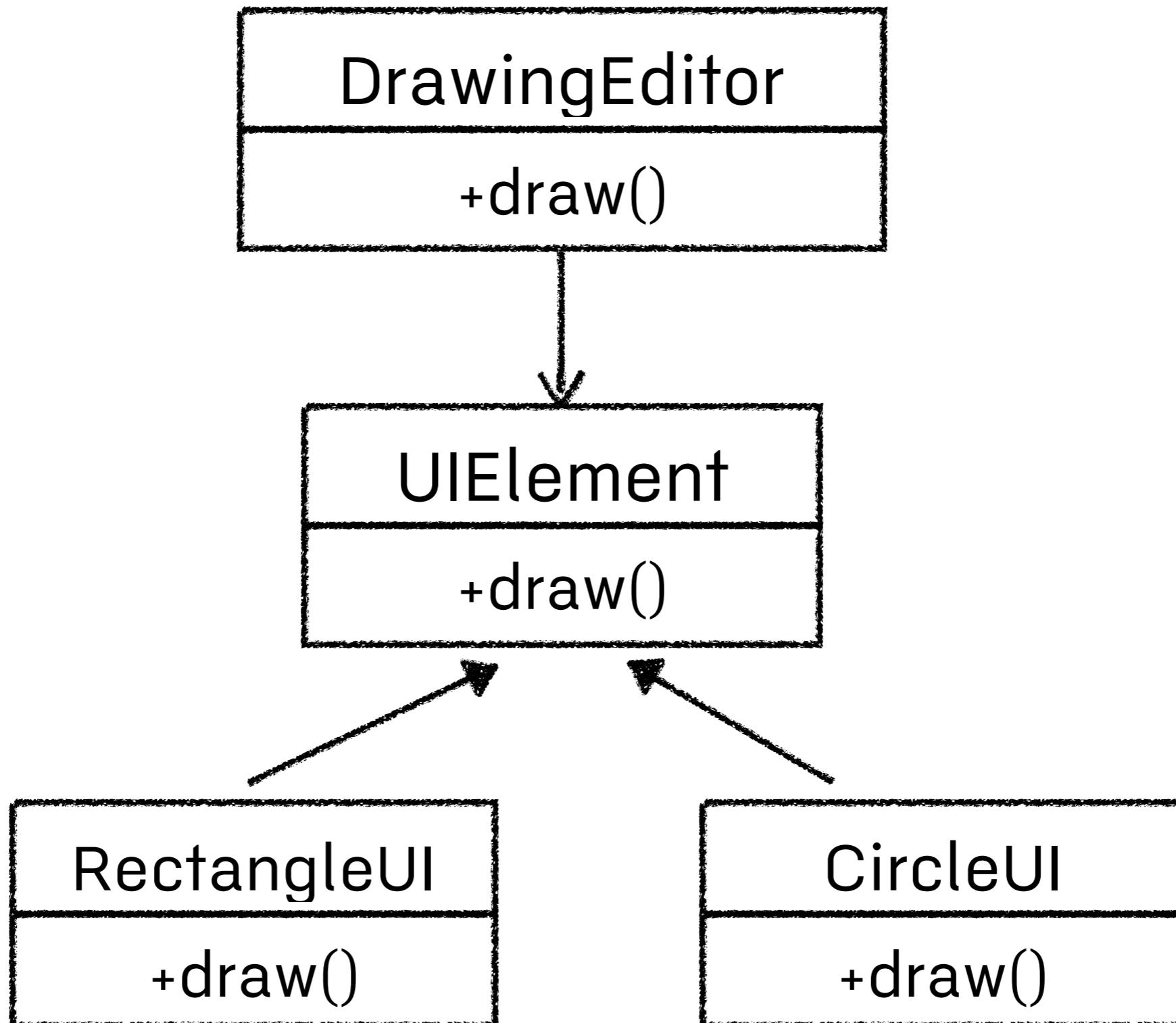
+draw()

-drawCircle()

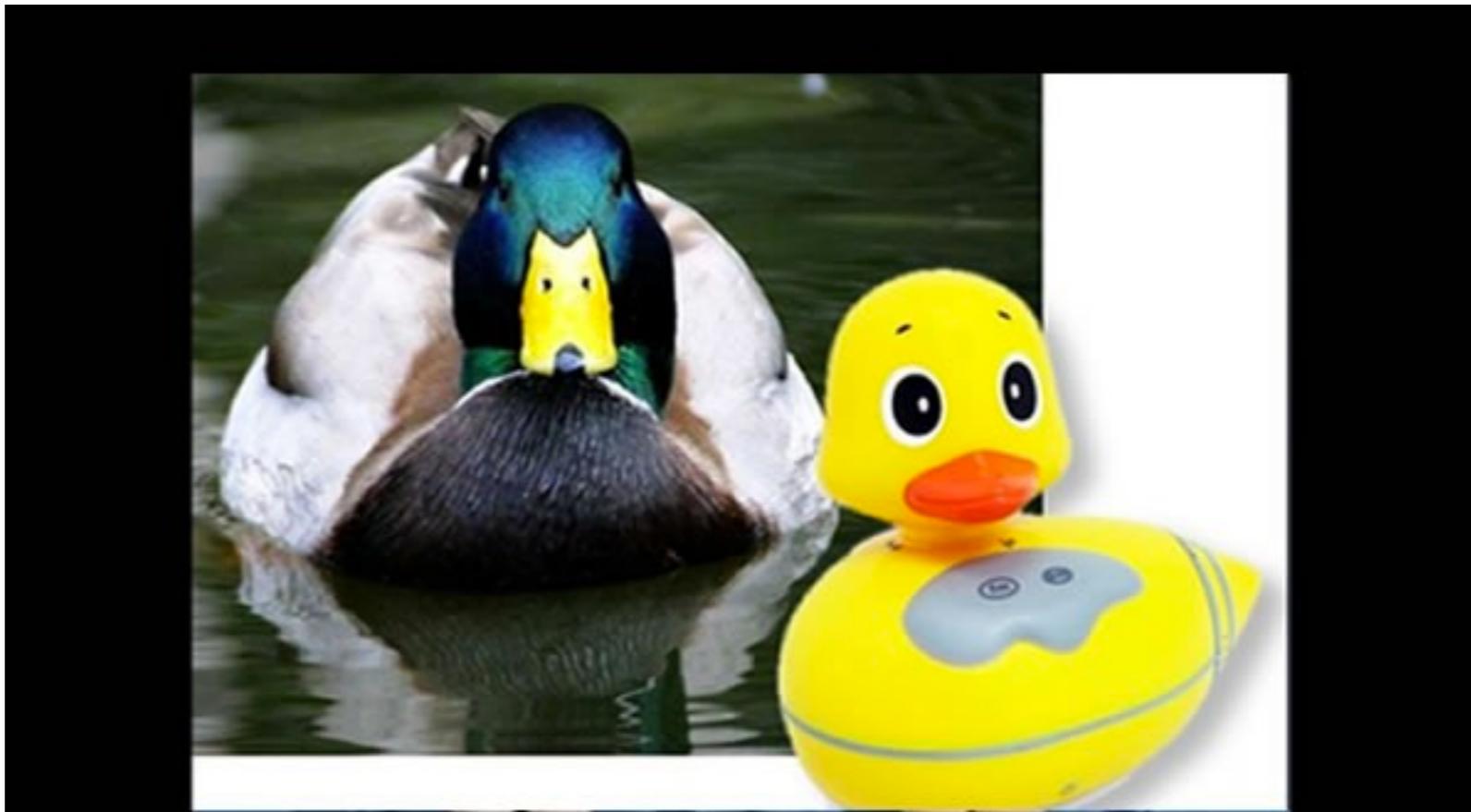
-drawRectangle()



# Example



# 3. Liskov Substitution Principle

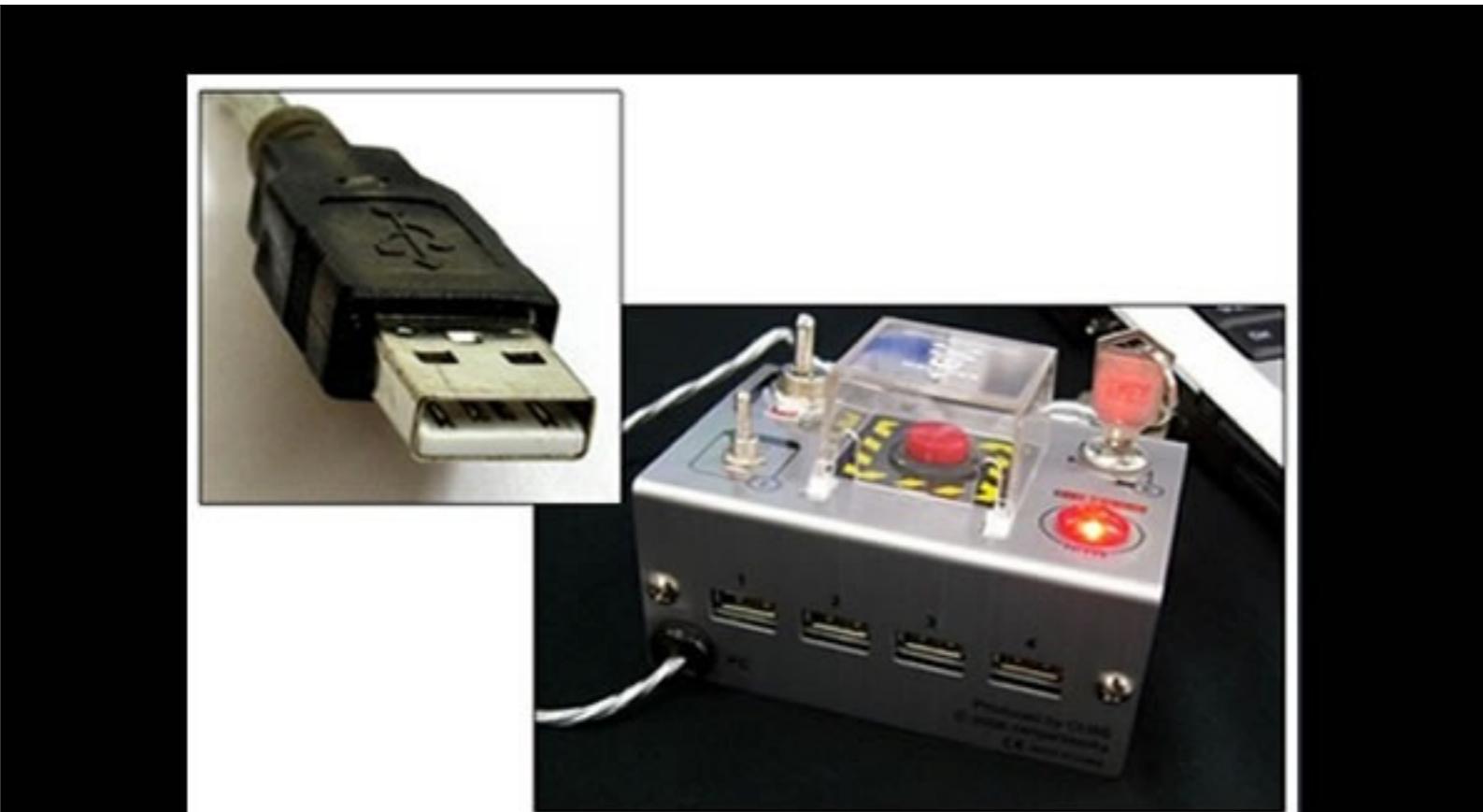


## Liskov Substitution Principle

If it looks like a duck and quacks like a duck but needs batteries,  
you probably have the wrong abstraction.



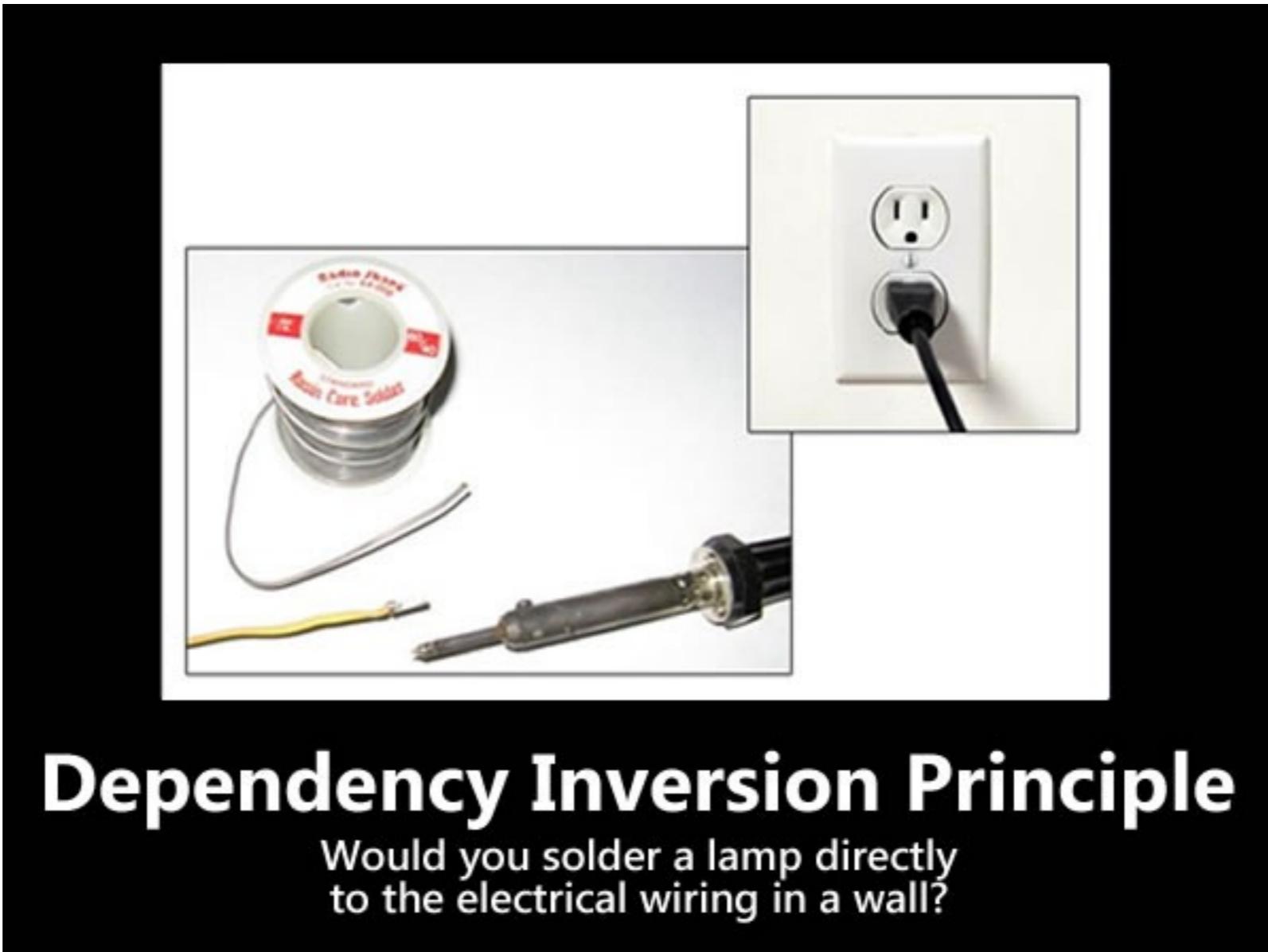
# 4. Interface Segregation Principle



**Interface Segregation Principle**  
You want me to plug this in *where?*



# 5. Dependency Inversion Principle



# Day 5



# Java Developer should learn 2018

<https://dzone.com/articles/5-things-java-programmer-should-learn-in-2018>



# Coding everyday (2 hours)



# Participate coding challenges



# Try new version of Java



# Learn Java performance tuning



# Learn Spring Framework



# Write Unit tests



# Profiling your app once a month



# Learn new JVM languages

