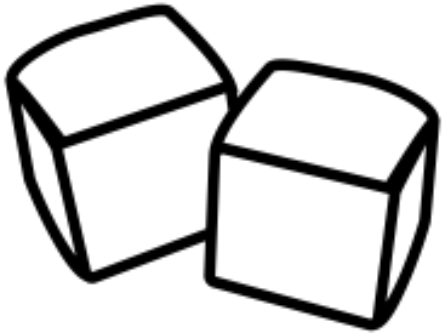
java 9 features

prepared & presented by cheng gibson @nossbigg


1
4
Days until Java 9 is officially released
Tweet this



but first, *sugar!*

1

JShell

>/ System.out.print(“Where’ve you been, Java?”)

JShell

```
public static void main(String[] args) {

}
```

```
40          result.add(number);  result:  size = 0  number: 1
41      }
42
43          retu
44  }
45
46  private
47      List
48      for
49
50      }
```

Evaluate Expression

number

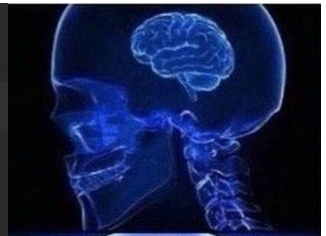
Use ⇧⌘⏎ to add to Watches

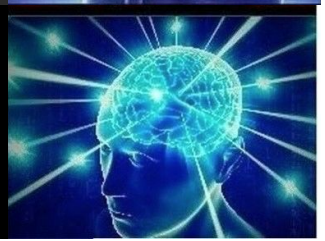
Result:

result = 1

```
sg:Contents gibson$ jshell
|  Welcome to JShell -- Version 9
|  For an introduction type: /help intro

jshell> "A shrewd fox lay in a bush of squirrels".matches(".*fox.*");
$1 ==> true
```





JShell

```
sg:Contents gibson$ jshell
|  Welcome to JShell -- Version 9
|  For an introduction type: /help intro

jshell> int a = 0;
a ==> 0

jshell> int a = 1;
a ==> 1

jshell> int b = 2;
b ==> 2

jshell> int add(int first, int second){return first+second;}
|  created method add(int,int)

jshell> add(a,b);
$5 ==> 3
```

A REPL Evaluator for Java!

Basic Commands

`/vars`	Displays all defined variables
`/methods`	Displays all defined methods
`/list`	Displays recent commands
`/edit <methodname>`	Opens an editor to edit a user-defined method

2

Convenience Factory Methods for Collections

~~int[] listOfInts = {1, 2, 3};~~

List.of(1,2,3);

Convenience Factory Methods for Collections

```java
private static List getListOfIntegersInJava4Way() {
    int[] listOfInts = {1, 2, 3};

    List result = new ArrayList();
    for (int index = 0; index < listOfInts.length; index++) {
        result.add(listOfInts[index]);
    }

    return result;
}
```

```java
private static List<Integer> getListOfIntegersInJava9Way() {
    return new ArrayList<>(List.of(1, 2, 3));
}
```

Convenience Factory Methods for Collections

- `List.of()`, `Set.of()`
- `Map.of()`, `Map.ofEntries()`
- Easy way to create lists/sets/maps
- **Immutable** objects

3

Improvements to Try-with-resources

Never use bufferedReader.close() again!

Improvements to Try-with-resources

They: Don’t worry, I never miss `close()` statements on my Reader objects!

Me: DoN’T wOrRy, i NeVeR MiSs `cLoSe()` sTaTeMenTs oN mY ReAdEr OBjEcTs!

Improvements to Try-with-resources

```java
BufferedReader reader1 = new BufferedReader(new FileReader(file1));
final BufferedReader reader2 = new BufferedReader(new FileReader(file2));

try (
        reader1; reader2;
        SampleAutoCloseable sampleAutoCloseable = new SampleAutoCloseable();
) {
    System.out.println(reader1.readLine());
    System.out.println(reader2.readLine());
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
} catch (Exception e) {
    e.printStackTrace();
```

Adds support for adding prior `Closeable` resources into `try-with-resources` block

4

Improvements to Optional class

Now with streams!

Improvements to Optional class

“I should use Optional here to warn developers that this method may return a null...”

“Nah, just throw an unchecked exception. They won’t notice”

Improvements to Optional class

```
List<String> firstNames = listOfOptionalPerson.stream()
        .flatMap(Optional::stream)
        .map(Person::getFirstName)
        .collect(Collectors.toList());

firstNames.forEach(System.out::println);
```

Optional: `.stream()`

Improvements to Optional class

```
Optional<String> availableName =
        Optional.ofNullable(person.getFirstName())
                .or(() -> Optional.ofNullable(person.getLastName()))
                .or(() -> Optional.ofNullable(person.getMiddleName()));
```

Optional: `.or()`

Improvements to Optional class

```
Optional.ofNullable(person.getFirstName())
        .ifPresentOrElse(
                System.out::println,
                () -> System.out.println("First name not found"));
```

Optional: `.ifPresentOrElse()`

5

Deprecation of Select Features

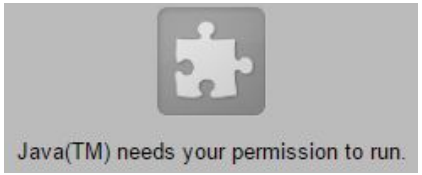
Out with the old...

Deprecation of Select Features

BLEACH, credits to Tite Kubo

Deprecation of Select Features

Deprecated Features



- ✖ Java Plug-In (used by Java Applets & JavaFX)

- ✖ ConcurrentMarkSweep Garbage Collector Strategy
- ✖ Some GC Strategy Combinations

and now,

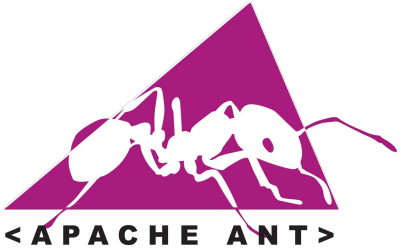
the heavy artillery

6

Java Platform Module System

Making code *even* more modular

Recap: Build Tools



- Find dependencies
- Download dependencies
- +some more stuff

Java Platform Module System

Pre-JPMS (< v1.9)

✖ Fat JRE/JDK

✖ Bloated Runtime Environments

✖ Brittle Classpath References

JPMS (v1.9)

✔ Lean JDK/JRE

✔ Lean Runtime Environments

✔ Module-Level References

¿ Solves JAR Hell

7

Reactive Streams

Asynchronous streams with backpressure

Reactive Streams

Elastic: The system stays responsive under varying workload. Reactive Systems can react to changes in the input rate by increasing or decreasing the resources allocated to service these inputs. This implies designs that have no contention points or central bottlenecks, resulting in the ability to shard or replicate components and distribute inputs among them. Reactive Systems support predictive, as well as Reactive, scaling algorithms by providing relevant live performance measures. They achieve elasticity in a cost-effective way on commodity hardware and software platforms.

Message Driven: Reactive Systems rely on asynchronous message-passing to establish a boundary between components that ensures loose coupling, isolation and location transparency. This boundary also provides the means to delegate failures as messages. Employing explicit message-passing enables load management, elasticity, and flow control by shaping and monitoring the message queues in the system and applying back-pressure when necessary. Location transparent messaging as a means of communication makes it possible for the management of failure to work with the same constructs and semantics across a cluster or within a single host. Non-blocking communication allows recipients to only consume resources while active, leading to less system overhead.

Responsive: The system responds in a timely manner if at all possible. Responsiveness is the cornerstone of usability and utility, but more than that, responsiveness means that problems may be detected quickly and dealt with effectively. Responsive systems focus on providing rapid and consistent response times, establishing reliable upper bounds so they deliver a consistent quality of service. This consistent behaviour in turn simplifies error handling, builds end user confidence, and encourages further interaction.

Resilient: The system stays responsive in the face of failure. This applies not only to highly-available, mission critical systems — any system that is not resilient will be unresponsive after a failure. Resilience is achieved by replication, containment, isolation and delegation. Failures are contained within each component, isolating components from each other and thereby ensuring that parts of the system can fail and recover without compromising the system as a whole. Recovery of each component is delegated to another (external) component and high-availability is ensured by replication where necessary. The client of a component is not burdened with handling its failures.

The Reactive Streams Manifesto

asynchronous stream processing

with

non-blocking backpressure

Reactive Streams

designed in collaboration with

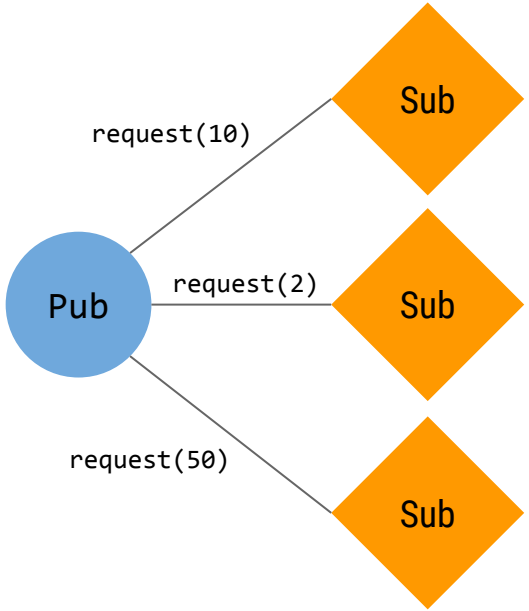

implemented by

Reactive Streams

Publish-Subscribe

+

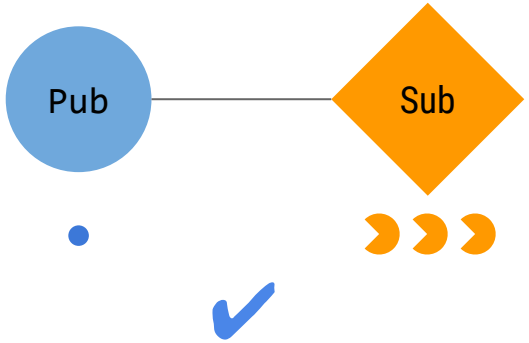
Rate-Limiting



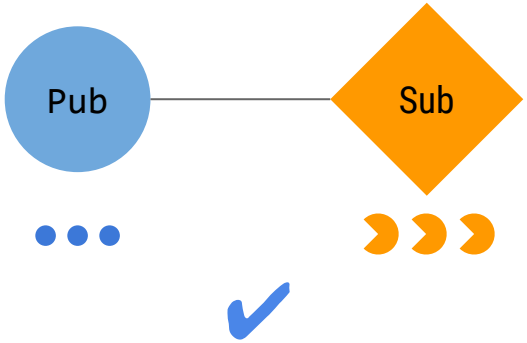
Reactive Streams

(push)

sub > pub

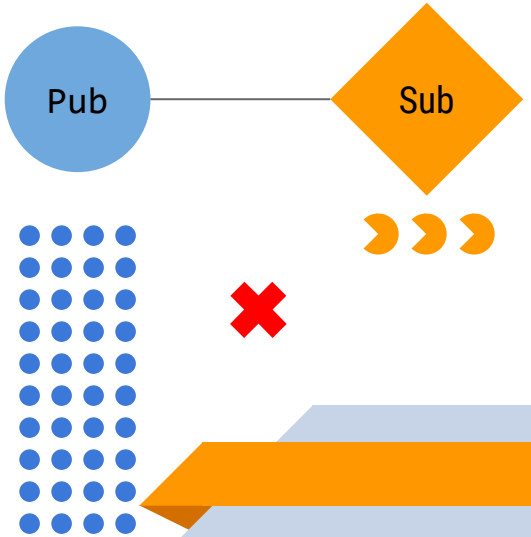


sub = pub



(pull)

sub < pub



java.util.concurrent.flow

The 4 Interfaces

Processor	Publisher	Subscriber	Subscription
	subscribe(Sub)	onComplete()	cancel()
		onError(Throw)	request(n)
		onNext(Item)	
		onSubscribe(Sub)	

8

Improvements to Nashorn

The JVM can run Javascript?
L(･o･)」

Recap: Nashorn

The JVM runs ECMAScript!

Improvements to Nashorn

Implements 6th Edition Updates to ECMAScript

- Keeping in pace with updates to ECMAScript

New Parser API

- Compiles ECMAScript into an AST
- *Particularly* useful for IDEs for code analysis

9

Other Updates

Because 9 ain’t enough. I know, I cheated ヾ(@^▽^@)ノ

Improvements to Nashorn

- G1 as Default Garbage Collection Strategy
- HTTP2 Client
- SHA-3 Hash Support
- HTML5 Support for Javadoc

Summary

Are you still awake? ^_^

Java Platform Module System	Reactive Streams	Improvements to Nashorn
JShell	Improvements to Optional class	Improvements for try-with-resources
Convenience Factor Methods for Collections	Deprecation of Select Features	Other Updates

FIN

Any questions?

chgibson@thoughtworks.com

CREDITS

Special thanks to all the people who made and released these awesome resources for free:

- Presentation template by SlidesCarnival
- Photographs by Startup Stock Photos