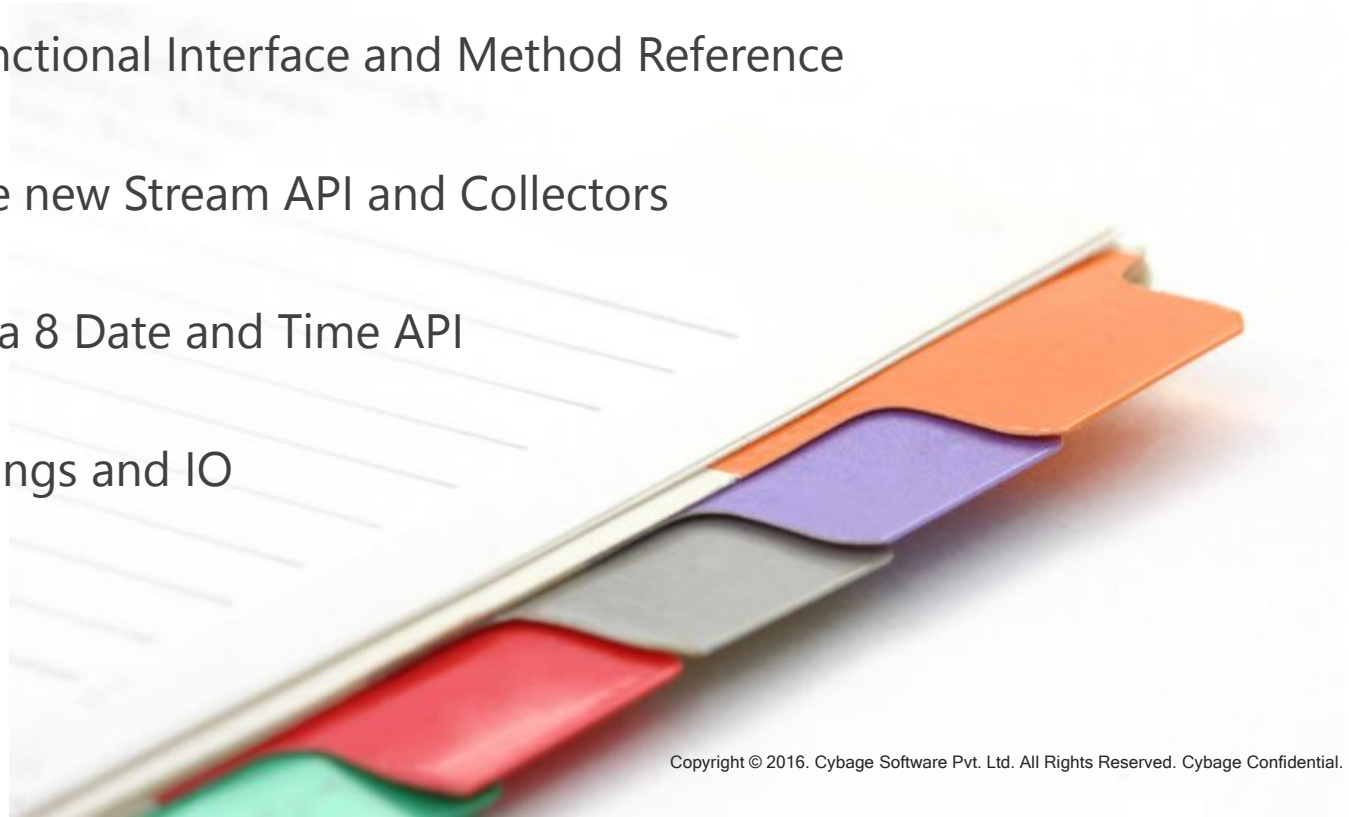
JAVA 8

Authored by :Asfiya Khan Presented by :Asfiya Khan

Cybage®
Delivering Value Scientifically

Agenda

- Introduction to new features of JAVA8
- Lambda Expressions in JAVA8
- Functional Interface and Method Reference
- The new Stream API and Collectors
- Java 8 Date and Time API
- Strings and IO



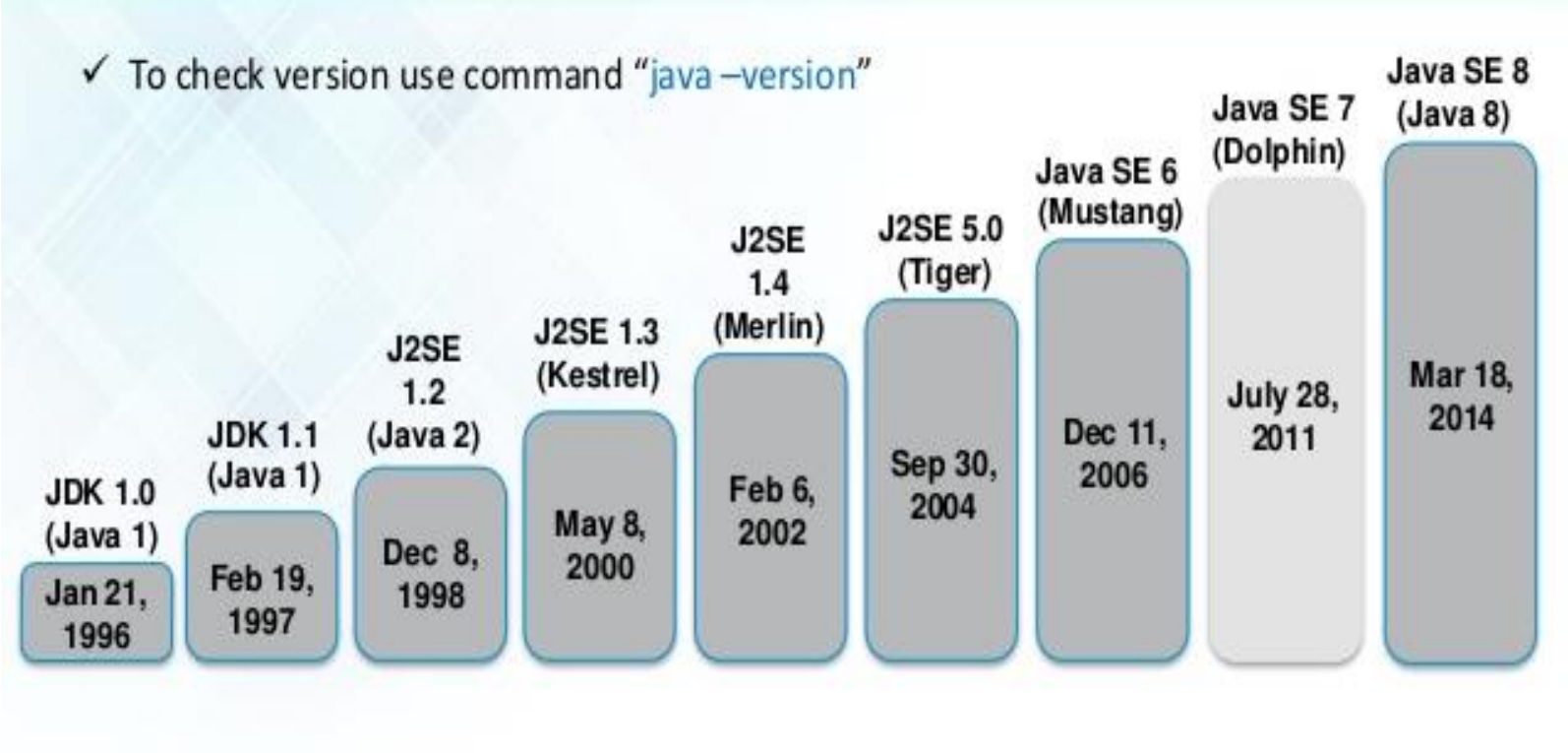
Introduction to New JAVA8

- Java 8 was a major release in terms of language and APIs.

- Includes several ideas from functional programming like :

i. behavior parameterization
ii. passing lambda expression as methods
iii. processing data with stream pipelines

Evolution of JAVA 8



New JAVA 8!!

LAMDA EXPRESSION

Lambda Expression

- It's an anonymous function which can be passed around in a concise way, or another way of writing instances of anonymous class.
- It comprises of:
  - set of parameters,
  - a lambda operator (->) and
  - a function body.

(set of parameters) -> function body;

Valid Lambda Expressions

```
n -> n % 2 != 0;
```

```
(char c) -> c == 'y';
```

```
(x, y) -> x + y;
```

```
(int a, int b) -> a * a + b * b;
```

```
() -> 42 () -> { return 3.14 };
```

```
(String s) -> { System.out.println(s); };
```

```
() -> { System.out.println("Hello World!"); };
```

Functional Interface

- An interface with exactly one abstract method becomes Functional Interface.
- @FunctionalInterface annotation is a facility to avoid accidental addition of abstract methods in the functional interfaces.

- A new package **java.util.function** has been added with bunch of functional interfaces to provide target types for lambda expressions and method references.

Valid Examples of Functional Interface

- ```
  public interface Runnable{
              run();
      };
  ```

- ```
  public interface Comparator<T>{
              int compare(T t1 ,T t2);
      };
  ```

- ```
  public interfaceFileFilter{
          boolean accept(File pathname);
    } ;
  ```
  ```

Functional Interface

- Functional Interface can be annotated and its optional.

```
@FunctionalInterface
public interface MyfunctionalInterface{
  someMethod();
 };
```

- Its just for the convenience , compiler can tell whether the interface is functional or not.

Four categories of Functional Interface

Inside Java.util.function we have 4 categories of interfaces:

1. Supplier

```
@FunctionalInterface
 public interface Supplier<T>{
   T get();
 };
```

It takes an object and provide a new object.

Categories of Functional Interface

2. Consumer/BiConsumer

```
@FunctionalInterface
      public interface Consumer<T>{
        void accept();
       };
```

Its reverse of Supplier, it accepts an object but doesn't return anything.

```
@FunctionalInterface
      public interface BiConsumer< T , U >{
        void accept( T  t , U u );
       };
```

This also accepts the object of different type.

Categories of Functional Interface

3. Predicate/BiPredicate

```
@FunctionalInterface
    public interface Predicate< T >{
      boolean test( T t );
    };
```

This method accept an Object and returns a boolean.

```
@FunctionalInterface
    public interface BiPredicate< T t , U u>{
      boolean test( T t , U u);
    };
```

This method accepts two objects of different type and returns a boolean.

Categories of Functional Interface

4. Function/BiFunction

```
@FunctionalInterface
      public interface Function < T , R >{
        R apply( T t );
       };
```

Represents a function that accept one arguments (T) and produces a result (R)

```
@FunctionalInterface
      public interface BiFunction < T , U , R >{
        R apply( T t ,U u );
       };
```

Represents a function that accepts two arguments (T , U) and produces a result (R)

Default Methods in Interface

- Java 8 introduces “Default Method” or (Defender methods) new feature, which allows Interface to define implementation to methods which will be used as default in the situation where a concrete Class fails to provide an implementation for that method.

```
public interface OldInterface {
    public void existingMethod();

    default public void newDefaultMethod() {
        System.out.println("New default method"
                + " is added in interface");
    }
}
```

What is a Stream?

An object on which one can define operations.

- A Stream **is** a pipeline of functions that can be evaluated
- Streams **can** transform data
- A Stream **is not** a data structure
- Streams **cannot** mutate data
- Stream is **not** a collection.

Stream is typed interface of type <T>

Why Streams?

1. In parallel to leverage the computing power of multicore CPU's.

2. In pipeline, to avoid unnecessary intermediary computations.

DEMO

Intermediate Operations

Function	Preserves count	Preserves type	Preserves order
map	✓	✗	✓
filter	✗	✓	✓
distinct	✗	✓	✓
sorted	✓	✓	✗
peek	✓	✓	✓

Terminal operations

Function	Output	When to use
reduce	concrete type	to cumulate elements
collect	list, map or set	to group elements
forEach	side effect	to perform a side effect on elements

Terminal Vs Intermediate Call

Terminal vs Intermediate Call

A terminal operation must be called to trigger the processing of a Stream

No terminal operation = no data is ever processed

Date and Time API

- New Date and Time API
- Instant and Duration
- Local date Period
- Temporal Adjusters
- LocalTime
- ZonedTime

Date Time API

Instant is a point on time line with nanosecond precision.

Duration is amount of time between two instant.

LocalDate to get the instance of date.

```
LocalDate.now()  //give the current date
```

Period is the amount of time between two LocalDate

LocalTime is a time of day.

Strings in Java 8

- **StringJoiner :** used to construct a sequence of characters separated by a delimiter and optionally starting with a supplied prefix and ending with a supplied suffix.

- **Streaming a String : chars()** creates a stream for all characters of the string

  eg -> "hey duke".chars().forEach(c -> System.out.println((char)c));

Any Questions?

Thank you!