Brighter Blue

Collections

Core Java Day 5

References

IBM

Brighter Blue

Contents

DEQUE INTERFACE AND TRANSFERQUEUE 4

STREAMS 7

PLUGGABLE ANNOTATIONS 9

IBM

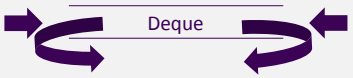
Deque Interface and TransferQueue

Slide Content	Notes

Deque Interface

The **Deque** interface extends **Queue** interface. It supports the concept of a double ended queue.

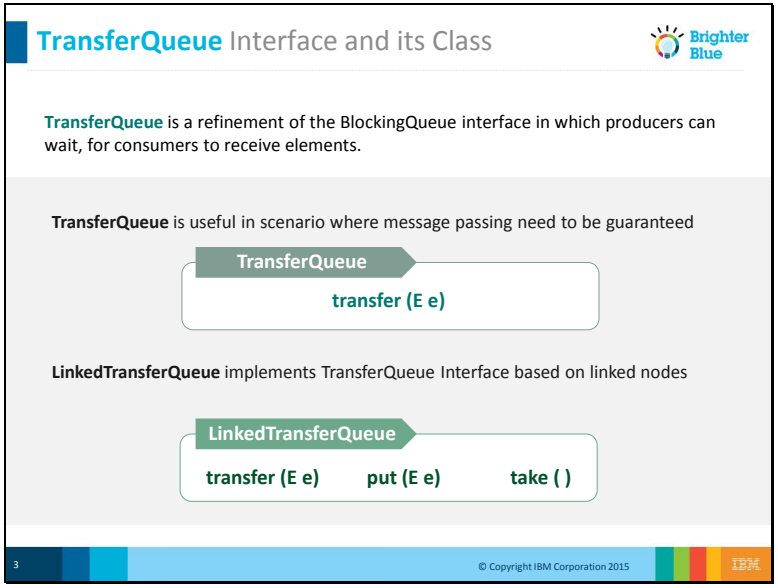
Double-ended queue supports the addition or removal of elements from either end of the data structure, it can be used as a queue (FIFO) or as a stack (LIFO).



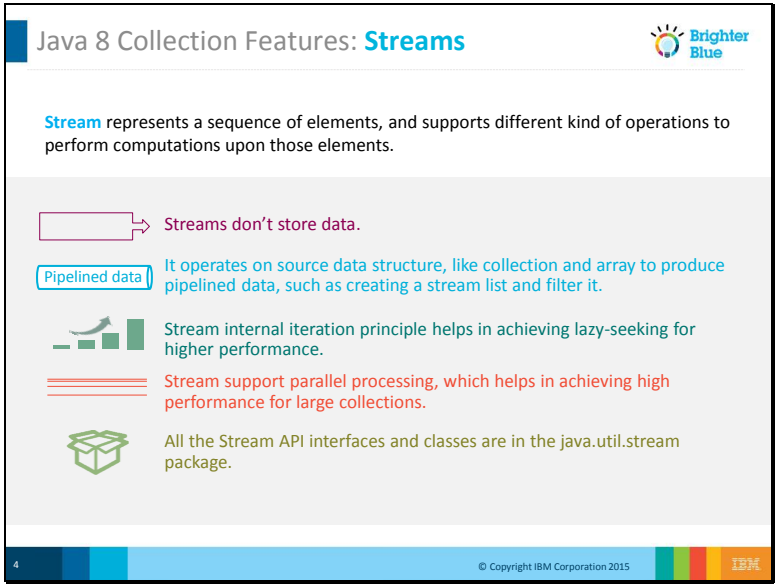
Operation	Special Value method	Exception throwing method
Insertion at head	offerFirst(e)	addFirst(e)
Removal at head	pollFirst()	removeFirst()
Retrieval at head	peekFirst()	getFirst()
Insertion at tail	offerLast(e)	addLast()
Removal at tail	pollLast()	removeLast(e)
Retrieval at tail	peekLast()	getLast()

1 © Copyright IBM Corporation 2015

Slide Content	Notes
## Sub Classes of **Deque** The three important sub classes of Deque Interface are : **ArrayDeque**, **BlockingDeque** and **LinkedBlockingDeque** **ArrayDeque** ▪ ArrayDeque is a class that implements the Deque. ▪ It has no capacity restrictions. ▪ It will perform faster than stack when used as stack, and faster than linked list when used as queue. ▪ ArrayDeque is not thread Safe. **BlockingDeque** ▪ When an element is inserted in a BlockingDeque, which is already full, it waits till the space becomes available and then insert an element. ▪ The time limit for waiting can be specified. **LinkedBlockingDeque** ▪ A LinkedBlockingDeque implements BlockingDeque interface, in which maximum capacity can be specified or **Integer.MAX_VALUE**. 2 © Copyright IBM Corporation 2015	

Slide Content	Notes
	The basic difference between TransferQueue and BlockingQueue is that TransferQueue waits for consumer to consume the element. TransferQueue makes it possible because producer will wait for consumer to transfer the message. The main method of TransferQueue is transfer(E e) . The LinkedTransferQueue is unbounded TransferQueue. It means LinkedTransferQueue has no size restriction. This queue follows first-in-first-out (FIFO) concept to get and add element. The important methods here are: ▪transfer(E e) : Inherited from TransferQueue in which producer transfers the element to consumer and waits if necessary. ▪put(E e) : producers put the element and it does not throw exception because there is no size restriction. LinkedTransferQueue is unbounded. ▪take() : It retrieves the element from queue and wait if empty.

Streams

Slide Content	Use this space for your own notes
	Collections are in-memory data structures, which hold elements within it. Each element in the collection is computed before it actually becomes a part of that collection. On the other hand Streams are fixed data structures, which computes the elements on-demand basis. The Java 8 Streams can be seen as lazily constructed Collections, where the values are computed when user demands for it. Stream doesn't store data, it operates on the source data structure (collection and array) and produce pipelined data that we can use and perform specific operations. Such as we can create a stream from the list and filter it based on a condition. Stream internal iteration principle helps in achieving lazy-seeking in some of the stream operations. For example filtering, mapping, or duplicate removal can be implemented lazily, allowing higher performance and scope for optimization. Stream support sequential as well as parallel processing, parallel processing can be very helpful in achieving high performance for large collections. All the Stream API interfaces and classes are in the java.util.stream

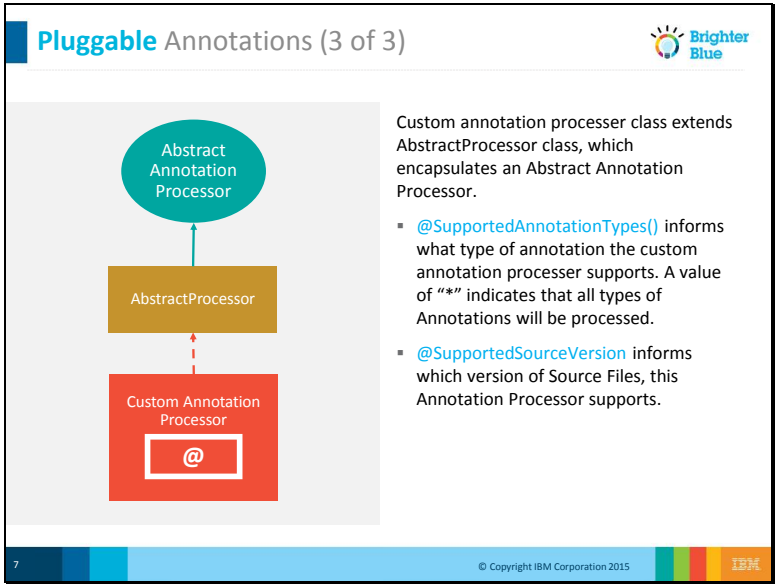
Slide Content	Use this space for your own notes
	package. Since we can use primitive data types such as int, long in the collections using auto-boxing and these operations could take a lot of time, there are specific classes for these – IntStream, LongStream and DoubleStream.

Pluggable Annotations

Slide Content	Use this space for your own notes
Pluggable Annotations (1 of 3) Brighter Blue **Pluggable Annotation** Processing API allows developers to write customized Annotation Processor, which can be plugged-in to the code to operate on the set of Annotations that appear in a Source File. The javac compiler of Mustang has an option called **'-processor'** where we can specify the Name of the Annotation Processor, along with a Set of Java Source Files containing the Annotations. **Syntax:** `javac -processor <Name of Annotation Processer> <Source files>` 5 © Copyright IBM Corporation 2015 IBM	Pluggable Annotation Processing API allows developers to write Customized Annotation Processor, which can be plugged-in to the code to operate on the set of Annotations that appear in a Source File.

Slide Content	Use this space for your own notes

Pluggable Annotations (2 of 3)
ClassLevelAnnotation.java
`package net.javabeat.articles.java6.newfeatures.customannotations;`
`import java.lang.annotation.*;`
`@Target(value = {ElementType.TYPE})`
`public @interface ClassLevelAnnotation { }`
MethodLevelAnnotation.java
`package net.javabeat.articles.java6.newfeatures.customannotations;`
`import java.lang.annotation.*;`
`@Target(value = {ElementType.METHOD})`
`public @interface MethodLevelAnnotation { }`
AnnotatedJavaFile.java
`package net.javabeat.articles.java6.newfeatures.customannotations;`
`@ClassLevelAnnotation()`
`public class AnnotatedJavaFile {`
`@MethodLevelAnnotation`
`public void annotatedMethod(){ }`
`}`
6 © Copyright IBM Corporation 2015
	Pluggable Annotation Processing API allows developers to write Customized Annotation Processor which can be plugged-in to the code to operate on the set of Annotations that appear in a Source File.
	
	Java Predefined Annotation :
	1. @Override : It is used when we override methods of superclass.
	2. @Deprecated : It is used when class or methods are deprecated.
	3. @SuppressWarnings : This is just to tell compiler to ignore specific warnings they produce.
	
	Some build in annotation is used to create custom annotation:
	1. @Documented : Javadoc will consider that element for documentation.
	2. @Inherited : Annotation type can be inherited from super class. If user put annotation type on class declaration but class declaration has no annotation for this type, then the classes superclass will automatically be queried for the annotation type.
	3. @Target : On which element you want to apply your annotation. If Target meta-annotation is not present, then annotation can be used on any program element. The values can be:
	ElementType.TYPE : Applies only to Type. A Type can be a Java class or interface or an Enum or even an Annotation.
	ElementType.FIELD : Applies to Java fields
	ElementType.METHOD : Applies to Java methods
	ElementType.PARAMETER : Applies only to method parameters in a

Slide Content	Use this space for your own notes
	method definition ElementType.CONSTRUCTOR : Applies only to a constructor of a class
	Pluggable Annotation Processing API allows developers to write Customized Annotation Processor, which can be plugged-in to the code to operate on the set of Annotations that appear in a Source File.