<u>Primitive data types & Wrapper classes</u>

Type	Description	Default	Size	Example	Wrapper Class
byte	whole number (-128 to 127)	0	8 bits	byte b = 12;	Byte
short	whole number (-32768 to 32767)	0	16 bits	short s = 123;	Short
int	whole number (-2^31 to 2^31-1)	0	32 bits	int i = 100;	Integer
long	whole number (-2^63 to 2^63-1)	0	64 bits	long l = 200;	Long
float	decimal number (1.4e-45 to 3.4e+38)	0.0	32 bits	float f = 1.3f;	Float
double	decimal number (4.94e-324 to 1.797e+308)	0.0	64 bits	double d = 0.5;	Double
char	'\u0000' to '\uffff'	'\u0000'	16 bits	char c = 'c';	Character
boolean	true / false	false	1 bit	boolean b = true;	Boolean

Abstract Class	**Interface**
IS-A relationship	HAS-A relationship
Can not create instance of an Abstract class	Can not create an instance of an Interface
Other classes should extend this Abstract class	Classes should implement the interface
0 or more abstract methods	By default all are abstract (From Java 8, Interfaces can have default and static non-abstract methods)
It can have private, protected, public members, methods	Can have static final variable. private members are not allowed. (From Java 9, interfaces can also have private methods)

Method References

Description	Lambda Expression	Lambda using Method Reference
Static method call - pass as parameter - Method accepts data and prints using System.out.println	(data) -> System.out.println(data)	System.out**::**println
Static method call - pass as parameter - Method accepts data and we pass it to another method to check if it is null	(o) -> Objects.isNull(o)	Objects::isNull
Given object -> Instance method call- for example, call the toUpperCase for the given string	(data) -> data.toUpperCase()	String::toUpperCase
Given object -> Instance method call with parameter- for example, call the concat for the given string	(s1, s2) -> s1.concat(s2)	String::concat
Given object -> Instance method call with parameters- for example, call the replaceAll for the given string with given parameters	(s1, s2, s3) -> s1.replaceAll(s2, s3)	String::replaceAll
Given object -> **pass as parameter** - pass the given object to another object method as parameter	(data) -> list.add(data)	list::add
Create new object -> new Cat()	() -> new Cat()	Cat::new

Java8 Functional Interfaces

Name	Input Type	Return Type	Method	Functional Composition / Lambda Chaining	Bi Type
Supplier<T>	N/A	T	get	N/A	N/A
Consumer<T>	T	void	accept	**andThen(Consumer<T>)**	BiConsumer<T, R>

Predicate<T>	T	boolean	test	**and(Predicate);**
				or(Predicate)
				negate()
					BiPredicate<T, R>

Function<T,R>	T	R	apply	**andThen(Function)**
				compose(Function)
					BiFunction<T,U, R>

Runnable	N/A	N/A	run	N/A	N/A
Callable<T>	N/A	T	call	N/A	N/A

Stream Operations

Type	Behavior	Methods
Intermediate	• Returns new streams • Lazy	• filter • map • limit • skip • distinct • sorted • flatMap • peek
Terminal	• Stream is consumed • Can not be reused	• forEach • collect • count • min • max • findAny • anyMatch • noneMatch

Intermediate Operations

Operation	Behavior	Input Data type	Example

filter
Intermediate Operations
	Used for filtering data	Predicate<T>	filter(i -> i % 2 == 0)
			filter(o -> Objects.nonNul(o))

map	Transforms the received data
	from one form to another
		Function<T, R>	map(i -> i * i)
			map(s -> s.toUpperCase())
			map(b -> DriverFactory.get(b))

limit	To limit the number of items which can flow through the pipeline	long	limit(3)
skip	Skips the first few items	long	skip(3)
peek	Just for debugging	Consumer<T>	peek(i -> System.out.println(i))
distinct	Allows only distinct values in the pipeline	N/A	distinct()

sorted	Sorts the data (asc / desc)	Comparator	sorted(Comparator.naturalOrder())
			sorted(Comparator.reverseOrder())

flatMap	Flattens the data	Function<T, R>	

Terminal Operations

Operation	Behavior	Input Data type	Example
forEach	Used for consuming the given object	Consumer<T>	forEach(e -> e.click())
collect	To collect the object into a list, set, map etc	Multiple implementations	collect(Collectors.toList())
count	To count the object	N/A	count()

min	first element after comparing all	Comparator	min(Comparator.naturalOrder())
			min(Comparator.reverseOrder())

max	last element after comparing all	Comparator	max(Comparator.naturalOrder())
			max(Comparator.reverseOrder())

findAny	Just give one from the stream	N/A	findAny()
findFirst()	Give the first one from the stream	N/A	findFirst()
anyMatch	is there any element in the stream which satisfies the condition?	Predicate<T>	anyMatch(i -> i > 5)
noneMatch	stream elements should not satisfy the given condition	Predicate<T>	noneMatch(i -> i > 5)

Terminal Operations - Collectors

Collect	Usage
To a list	.collect(Collectors.toList())
To a set (no duplicates)	.collect(Collectors.toSet())

Join all	.collect(Collectors.joining())
	.collect(Collectors.joining(","))

To a map	.collect(Collectors.groupingBy(...))

Stream Source

Source	Usage
List list	list.stream()
Set set	set.stream()

Map map	map.entrySet().stream()
	map.keySet().stream()
	map.values().stream()

int[] arr	Arrays.stream(arr)
String a = "udemy" String b = "hi" String c = "hello"	Stream.of(a, b, c.....)

Comparator

Comparator	Usage

Comparator.naturalOrder()	min(Comparator.naturalOrder())
	max(Comparator.naturalOrder())
	sorted(Comparator.naturalOrder())

Comparator.reverseOrder()	min(Comparator.reverseOrder())
	max(Comparator.reverseOrder())
	sorted(Comparator.reverseOrder())

Comparator.comparing(Function)	//Student name
	min(Comparator.comparing(s -> s.getName()))

Primitive Streams

Stream<T>	Convert	Primitive Streams	Terminal Operations

Stream<Integer>	mapToInt(Function) ==>
	
	<== boxed()
		IntStream	sum()
			average()

Stream<Long>	mapToLong(Function) ==>
	
	<== boxed()
		LongStream	sum()
			average()

Stream<Double>	mapToDouble(Function) ==>
	
	<== boxed()
		DoubleStream	sum()
			average()