Generics in Java

Basics

Raw Types : Collections that done have a generic on them
List<String> is not subclass of List<Object>

Unbounded Wildcards : ? operator (implies we dont knw the underlying type)

List<?> : read from it is possible but writing into the list is not possible because the add method doesnt know the type

List<?> studd = Arrays.asList("asas",1223);
stuff.size() --- returns 2
stuff.forEach((Object o) -> println(o)) --- prints the element of stuff
stuff.add("asdas") ----- doesnt work because add needs a parameter type but the ? doesnt specify any type.

UpperBound on a wildcard

Allows us to tell compiler what type we are expecting.
? extends MyClass ---→ you can read, and define but not add elements. the bound should be MyClass or one of its SubClass

List<? extends Number> numbers = new ArrayList<>();
numbers.add(bvalue) → doesnt compile

List<? extends Number> can take any type which is subclass of Number

LowerBound on a wildcard

? super MyClass : the bound should be atleast MyClass or one of its SuperClass

PECS : Producer Extends, Consumer Super

Use extends when you consume
use super when you produce
use both if you are doing both

Java 8 Streams : provide a value , that uses Super, consuming a stream value then use Extends

Stream.max

Optional<T> max (Comparator<? super T> comparator)

? super T indicates the input could be T or any super class of T.

Employee maxId - employees.stream().max(comparingInt(Employee :: getId)).orElse (new Employee(121,"sasa"))

comparing(Function<? super T, ? extends U> keyExtractor)

? super T indicates that the value is being extracted from stream
? extends U indicates that the value is being produced by the method as output

Stream.map

<R> Stream <R> map(Function< ? super T, ? extends R>) mapper)

Collectors

Collectors.toMap

static <T,K,U> Collector <T, ? , Map<K,U> > toMap (
Function<? super T, ? extends K> keyMapper,
Function<? super T, ? extends U> valueMapper)

return type is Collector <T,A,R>
T- input element type
A- mutable accumulation list for intermediate reduction
R- Reduced data type

Type Erasure

Generics exist and enforced at compile time but not present at RunTime.
Erasure of data type occurs

Bounded type Params are replaced by their Bound
Unbounded type Params are replaced by Object
Type casts areinserted as and when needed
Bridge methods to ensure polymorphism

Eg:

```
public class Node <T>{
    private T data;
    private Node<T> next;

    //getters and setters
}
```

converted to after compile

```
public class Node{
    private Object data;
    private Node next;

    //getters and setters
}
```

Eg:

```
public class Node <T extends Comparable>{
    private T data;
    private Node<T> next;

    //getters and setters
}


converted to after compile

public class Node{
    private Comparable data;
    private Node next;

    //getters and setters
}
```