How to run a java app using command prompt

1. Compile it and a .class file will be available

A screenshot of a computer

Description automatically generated

1. In the above pic, the .class file is in the “out” folder
2. Open command prompt to where the out folder is

A screen shot of a computer

Description automatically generated

A black screen with white text

Description automatically generated

1. What happens when you run the “java” command in the command prompt?
2. The bytecode from the .class file is loaded into memory.
3. JVM looks for a main method
4. To get the JVM, you need to download the JDK, java development kit, which includes the JVM !
5. JDK is for developing java programs, JRE is only to run them.

57 reserved keywords as of java 17

2 examples are if and else

public class FirstClass {

public static void main(String[] args) {

for (int i =0; i<10; i++) {

System.out.println("nischal " + i);

}

if (2 > 100) {

System.out.println("test");

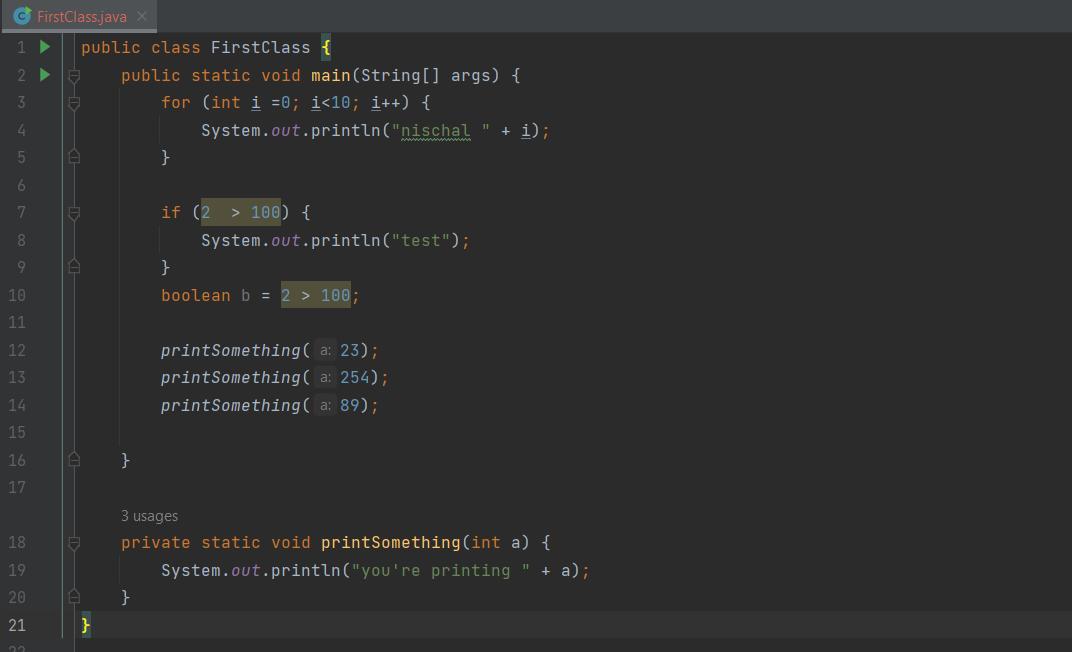
}

}

}

In the above, if is a keyword, 2>100 is an expression

An expression is something that can be resolved to a value



printSomething is a method

* Methods allow you to code reuse
* Change code at only one point.

A screen shot of a computer program

Description automatically generated

Java chooses it at compile time.

A screenshot of a computer program

Description automatically generated

If you don’t have break, then the program will flow into the next case

Until a break statement is encountered.

New switch

A screenshot of a computer program

Description automatically generated

Enhanced switch

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

For loop

A screenshot of a computer program

Description automatically generated

While loop

A screenshot of a computer program

Description automatically generated

Do while loop

A screenshot of a computer

Description automatically generated

It will execute the loop at least once.

A screenshot of a computer

Description automatically generated

Above represents a local variable and code block

Variables should be declared in the smallest possible scope

A screenshot of a computer program

Description automatically generated

Static methods are used to access static variables – this makes sense because you cant call an instance variable – the static method wont know “which one” to choose as instance variables depends on the instance.

Instance methods can access static variables – this also makes sense

A screenshot of a computer program

Description automatically generated

This is exception handling.

Inheritance

Slide 76:

C:\TimBuchalkaCourse\slide76

A screen shot of a computer

Description automatically generated

public class MainCar {  
 public static void main(String[] args) {  
 Car c = new Car();  
 c.describeCar();  
 }  
}

here, you will get all default values for the variables which are 0, null, null, null , “”

convertible is false by default!

These you can add getters

These you can add setters

A screenshot of a computer program

Description automatically generated

The setMake method is an example of encapsulation.

You don’t allow the make to be changed directly by the client, so you make it private

And you have setter method that dictates how the value of make should be set.

A screen shot of a computer

Description automatically generated

Above is an example of a constructor

If you create account object with default constructor, then you will have all default values for the instance variables.

If you create a constructor of your own, then java wont automatically create the default constructor for you. So if you removed the empty constructor in the above pic, then calling a default constructor from the main class will be a compile error.

A screenshot of a computer program

Description automatically generated

You can use this from a constructor to call another constructor!

Reference vs instance vs class

A screenshot of a computer program

Description automatically generated

Aero is a class

A is an instance of the Aero class

The reference of the instance of the Aero class is passed to changeA();

The pojo

public class ReferenceVsInstanceVsClass {  
 public static void main(String[] args) {  
  
 Boy b = new Boy("nish",28);  
 System.*out*.println(b.age());  
 System.*out*.println(b.name());  
 }  
}  
  
record Boy(String name, int age) {  
  
}

the record is a pojo without any setters – it is immutable!

Inheritance 1

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

Dog takes the move, toString and makenoise methods from Animal

However, dog’s variables are different, because it calls Animal’s constructor and sets the values of its own.

Every class in java inherits from Java.Lang.Object

A screenshot of a computer program

Description automatically generated

The @Override annotation suggests that toString() is coming from a parent class – in this case the object class.

This vs super

This is used to call a constructor in the same class.

Super is used to call a constructor of a parent class.

Method overloading – happens to methods within the same class and the method chosen can be decided by compile time by java

Method overriding – the overloaded method to be called is decided at runtime

A screen shot of a computer

Description automatically generated

Above is the string pool concept

Both s and r point to the same object in the string pool

A screen shot of a computer

Description automatically generated

Strings are immutable!

A screenshot of a computer program

Description automatically generated

If you want to manipulate Strings, use StringBuilder. – because this avoids creating intermediate String objects !

A screen shot of a computer

Description automatically generated

Above is an example of composition

PersonalComputer “HAS-A” monitor

You can use a personalComputer object to call the monitor object which it contains and execute the monitor’s functions

Polymorphism

A screenshot of a computer program

Description automatically generated

At runtime, java is able to decide which concrete instance type the reference is pointing to, and will accordingly run the methods !

A computer screen shot of text

Description automatically generated

When the program is running, you can assign the reference variable of type Movie to any of its subclass concretes. At RUNTIME

Then also at runtime java can decide which particular implementation to call based on the type of the concrete child.

A screenshot of a computer program

Description automatically generated

Arrays

public class InstanceOfMain {  
 public static void main(String[] args) {  
 int arr[] = {7,6,2,3,1};  
 System.*out*.println(arr[3]);  
 }  
}

time complexity of this program is

for the accessing operation is O(1)

arrays are stored in contiguous memory locations

[7][6][2][3][1]

Because of this continugous nature, java can predict and know where to retrieve the element from, and hence no need to traverse the array to find the element.

ArrayLists internally are implemented using arrays

Therefore, if you want to access, it is O(1) time complexity

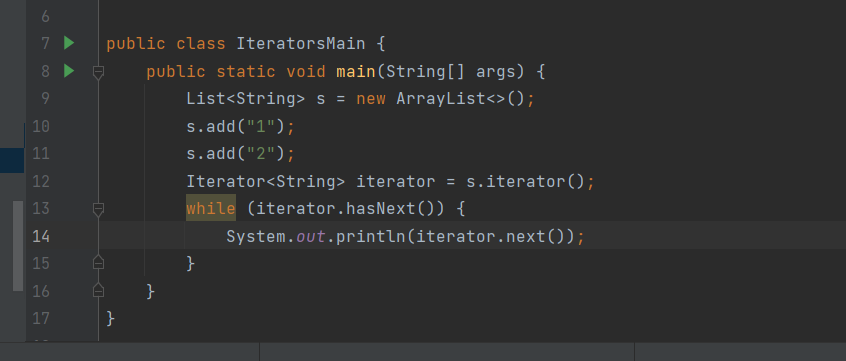
However, if you want to add an element in the centre for example, then it is not so efficient, because you have to shift the elements of the array

LinkedList

Insert and remove element is easier than arraylist

If you are doing operations where a large amount of data is being manipulated lets say from the middle of the list all the time, then use linkedlists

Iterators



Autoboxing vs unboxing

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

package emi;  
public class EnumMain {  
 public static void main(String[] args) {  
 DaysOfWeek[] d = DaysOfWeek.*values*();  
 for (int i =0;i<d.length; i++) {  
 System.*out*.println(d[i].ordinal());  
 System.*out*.println(d[i].name());  
 System.*out*.println(d[i].getValore());  
 System.*out*.println(d[i].getBroke());  
 }  
 }  
}  
enum DaysOfWeek {  
 *Sunday*(1,"hi"),  
 *Monday*(2,"hey!");  
 private final int valore;  
 private String broke;  
 DaysOfWeek(int valore,String broke) {  
 this.valore = valore;  
 this.broke = broke;  
 }  
 public int getValore() {  
 return valore;  
 }  
  
 public String getBroke() {  
 return broke;  
 }  
}

When to use abstract classes in java

package animal;  
  
public abstract class Animal {  
  
 protected String type;  
 private String size;  
 private double weight;  
  
 public Animal(String type, String size, double weight) {  
 this.type = type;  
 this.size = size;  
 this.weight = weight;  
 }  
  
 public abstract void move(String speed);  
 public abstract void makeNoise();  
  
}

package animal;  
  
public class Dog extends Animal {  
  
 public Dog(String type, String size, double weight) {  
 super(type, size, weight);  
 }  
  
 @Override  
 public void move(String speed) {  
  
 System.*out*.println("doge moving");  
 }  
  
 @Override  
 public void makeNoise() {  
 System.*out*.println("doge making noise!");  
  
 }  
}

you can see that Animal on its own doesn’t make sense so you block it from being implemented

however it shares a lot of similar features with dog

use abstract class when class A “is – A” B

use interface when class A “is – a – type – of “ B

Interface

public interface Flyable {  
 public String howDoesItFly();  
}  
  
class Bird implements Flyable {  
  
 @Override  
 public String howDoesItFly() {  
 return null;  
 }  
}  
  
class Plane implements Flyable {  
  
 @Override  
 public String howDoesItFly() {  
 return null;  
 }  
}

public interface Flyable {  
 public String howDoesItFly();  
 default void doSomething() {  
 System.*out*.println("doing!");  
 }  
}  
  
class Bird implements Flyable {  
  
 @Override  
 public String howDoesItFly() {  
 return null;  
 }  
   
 public void doSomething() {  
 System.*out*.println("bird dpomg something");  
 }  
}  
  
class Plane implements Flyable {  
  
 @Override  
 public String howDoesItFly() {  
 return null;  
 }  
}

default is for backward compatibility

lets say u want to add a method to an interface, and that interface has 10000 implementations

adding a default method will ensure that only those 1 or 2 implementations that need to override this default will be affected and you wont need to go do something weird like add an empty method body in the rest

import java.util.ArrayList;  
import java.util.Collections;  
import java.util.Comparator;  
import java.util.List;  
  
public class ComparatorPrac {  
 public static void main(String[] args) {  
 Student s = new Student();  
 s.name = "manu";  
 s.age = 23;  
  
 Student s1 = new Student();  
 s1.name = "manukanta";  
 s1.age = 21;  
  
 List<Student> students = new ArrayList<>();  
 students.add(s); students.add(s1);  
  
 Collections.*sort*(students,new StudentAgeComparator());  
  
 System.*out*.println(students);  
  
 for (Student s3 : students) {  
 System.*out*.println(s3.name);  
 }  
  
 }  
}  
class Student {  
 public String name;  
 public int age;  
}  
  
class StudentAgeComparator implements Comparator<Student> {  
  
 @Override  
 public int compare(Student o1, Student o2) {  
 return o1.age - o2.age;  
 }  
}

package animal;  
  
public class GenericExample {  
 public static void main(String[] args) {  
 Secontion<String> s = new Secontion<>("duplex");  
 Secontion<Integer> i = new Secontion<>(23);  
  
 System.*out*.println(s.getVariabel().toUpperCase());  
 System.*out*.println(i.getVariabel().byteValue());  
  
 }  
}  
  
class Secontion <T> {  
 T variable;  
  
 public Secontion(T r) {  
 this.variable = r;  
 }  
  
 public T getVariabel() {  
 return variable;  
 }  
}

use generics when the difference between 2 classes is only the datatype

the methods, behaviour all the same, then use generics!

If you want to reuse code and make your class more flexible

If you want to ensure type safety

package innerclass;  
  
public class UseInnerClass {  
 public static void main(String[] args) {  
 AirCon.Inner a = new AirCon.Inner();  
 System.*out*.println(a.getOuterName());  
 System.*out*.println(a.getInnerName());  
 }  
}  
  
class AirCon {  
  
 private static String *name* = "darius";  
  
 public static class Inner {  
  
 private String name2 = "nish";  
  
 public String getOuterName() {  
 return *name*;  
 }  
  
 public String getInnerName() {  
 return this.name2;  
 }  
  
 }  
}

When do you use a static inner class?

Grouping classes together

Eg. You have a person class, and inside it u have an Address class

package anoninner;  
  
interface nish {  
 public String test(String x);  
}  
  
public class MainSettle {  
  
 static nish *a* = new nish() {  
  
 @Override  
 public String test(String x) {  
 return x.toUpperCase();  
 }  
 };  
  
 public static void main(String[] args) {  
 String jeef = *a*.test("jeffrey");  
 System.*out*.println(jeef);  
 }  
}

@FunctionalInterface  
public interface SingleInstanceFun {  
 public String upperCaseIt(String x);  
}

@FunctionalInterface  
 interface SingleInstanceFun3 {  
 public String upperCaseIt(String x);  
}  
  
public class FunIntMain {  
 public static void main(String[] args) {  
 SingleInstanceFun3 singleInstanceFun3  
 = String::toUpperCase;  
  
 SingleInstanceFun3 singleInstanceFun4  
 = x -> x.toUpperCase();  
  
 System.*out*.println(*getSomething*(singleInstanceFun3,"jim"));  
 System.*out*.println(*getSomething*(singleInstanceFun4,"carrey"));  
  
 }  
  
 private static String getSomething(SingleInstanceFun3 singleInstanceFun3,String x) {  
 return singleInstanceFun3.upperCaseIt(x);  
 }  
}

Streams

package stremz;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.stream.Collectors;  
  
public class StreamMain {  
 public static void main(String[] args) {  
  
 //consumer  
 //predicate  
 //supplier  
 //function  
  
 List<String> strings = new ArrayList<>();  
  
 strings.add("a");  
 strings.add("b");  
 strings.add("c");  
 strings.add("d");  
 strings.add("e");  
 strings.add("f");  
 strings.add("g");  
 strings.add("h");  
  
 List<String> collect = strings.stream().limit(3)  
 .filter(a -> a.equals("a"))  
 .map(String::toUpperCase).collect(Collectors.*toList*());  
 for (String a : collect) {  
 System.*out*.println(a);  
 }  
  
 System.*out*.println("---------");  
 for (String a : strings) {  
 System.*out*.println(a);  
 }  
 }  
}

in the above .collect is a terminal operator

stream can only be used once –

Streams practice

package stremz;  
  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.List;  
import java.util.Optional;  
import java.util.stream.Collectors;  
  
public class StreamMain {  
 public static void main(String[] args) {  
  
 //consumer  
 //predicate  
 //supplier  
 //function  
  
 List<String> strings = new ArrayList<>();  
  
 strings.add("a");  
 strings.add("b");  
 strings.add("c");  
 strings.add("d");  
 strings.add("d");  
 strings.add("f");  
 strings.add("g");  
 strings.add("h");  
  
 List<String> strings1 = strings.stream()  
 .distinct()  
 .toList();  
 System.*out*.println(strings1);  
  
 List<String> strings2 = strings.stream()  
 .filter(x -> !x.equals("d"))  
 .toList();  
 System.*out*.println(strings2);  
  
 List<String> strings3 = strings.stream()  
 .skip(4)  
 .toList();  
 System.*out*.println(strings3);  
  
  
  
 List<String> strings4 = strings.stream()  
 .skip(4)  
 .sorted()  
 .toList();  
 System.*out*.println(strings4);  
 //System.out.println(strings3);  
  
 List<String> strings5 = strings.stream()  
 .map(x -> x.toUpperCase())  
 .toList();  
  
 System.*out*.println(strings5);  
  
 List<Integer> x = Arrays.*asList*(1,7,8,6,4,5);  
  
 List<Integer> strings6 = x.stream()  
 .filter(StreamMain::*isEven*)  
 .toList();  
  
 System.*out*.println(strings6);  
  
 Optional<Integer> reduce = x.stream().reduce((x8, x9) -> x8 + x9);  
  
 Integer integer = reduce.get();  
  
 System.*out*.println(integer);  
  
 System.*out*.println(strings);  
 }  
  
 private static boolean isEven(int x) {  
 if (x % 2 == 0) {  
 return true;  
 }else {  
 return false;  
 }  
 }  
  
}

Optional

package stremz;  
  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.List;  
import java.util.Optional;  
import java.util.stream.Collectors;  
  
public class StreamMain {  
 public static void main(String[] args) {  
  
 Optional<String> even = *isEven*(2);  
 Optional<String> even1 = *isEven*(3);  
  
 System.*out*.println(even == null);  
 System.*out*.println(even1 == null );  
  
 if (even.isPresent()) {  
 String s = even.get();  
 System.*out*.println(s);  
 }  
  
 if (even1.isPresent()) {  
 System.*out*.println(even1.get());  
 } else {  
 System.*out*.println("something went wrong");  
 }  
  
 String s = even.orElse("jim!");  
 String s1 = even1.orElseGet(() ->"heyyabro!");  
  
 System.*out*.println(s);  
 System.*out*.println(s1);  
 }  
  
 private static Optional<String> isEven(int x) {  
 if (x == 2) {  
 return Optional.*of*("hi!");  
 }else {  
 return Optional.*empty*();  
 }  
 }  
  
}

package stremz;  
  
import java.util.\*;  
import java.util.stream.Collectors;  
  
public class StreamMain {  
 public static void main(String[] args) {  
  
 List<Student> students = new ArrayList<>();  
 students.add(new Student("A","1234"));  
 students.add(new Student("B","1235"));  
 students.add(new Student("A","1236"));  
 students.add(new Student("D","1237"));  
 students.add(new Student("A","1238"));  
  
 Map<String, List<Student>> collect = students.stream().collect(Collectors.*groupingBy*(student -> student.getClazz()));  
  
 for (Map.Entry<String, List<Student>> entry : collect.entrySet()) {  
 System.*out*.println("Key: " + entry.getKey() + ", Value: " + entry.getValue());  
 }  
 }  
  
}  
  
class Student {  
 private String clazz;  
 private String id;  
  
 public Student(String clazz, String id) {  
 this.clazz = clazz;  
 this.id = id;  
 }  
  
 public String getClazz() {  
 return clazz;  
 }  
  
 public void setClazz(String clazz) {  
 this.clazz = clazz;  
 }  
  
 public String getId() {  
 return id;  
 }  
  
 public void setId(String id) {  
 this.id = id;  
 }  
}

package stremz;  
  
import java.util.\*;  
import java.util.stream.Collectors;  
  
public class StreamMain {  
 public static void main(String[] args) {  
  
 List<Student> students = new ArrayList<>();  
 students.add(new Student("A","1234"));  
 students.add(new Student("B","1235"));  
 students.add(new Student("A","1236"));  
 students.add(new Student("D","1237"));  
 students.add(new Student("A","1238"));  
  
 Map<String, List<Student>> collect = students.stream().collect(Collectors.*groupingBy*(student -> student.getClazz()));  
  
 for (Map.Entry<String, List<Student>> entry : collect.entrySet()) {  
 System.*out*.println("Key: " + entry.getKey() + ", Value: " + entry.getValue());  
 }  
  
 Map<String, List<Student>> collect2 = students.stream().collect(Collectors.*groupingBy*(student -> student.getClazz() + student.getId()));  
  
 for (Map.Entry<String, List<Student>> entry : collect2.entrySet()) {  
 System.*out*.println("Key: " + entry.getKey() + ", Value: " + entry.getValue());  
 }  
 }  
  
}  
  
class Student {  
 private String clazz;  
 private String id;  
  
 public Student(String clazz, String id) {  
 this.clazz = clazz;  
 this.id = id;  
 }  
  
 public String getClazz() {  
 return clazz;  
 }  
  
 public void setClazz(String clazz) {  
 this.clazz = clazz;  
 }  
  
 public String getId() {  
 return id;  
 }  
  
 public void setId(String id) {  
 this.id = id;  
 }  
}

package stremz;  
  
import java.util.\*;  
import java.util.stream.Collectors;  
  
public class StreamMain {  
 public static void main(String[] args) {  
  
 List<Integer> numbers = Arrays.*asList*(2815,5678,2234);  
 Student a = new Student("A",numbers);  
  
 List<Integer> numbers2 = Arrays.*asList*(1111,2222,3333);  
 Student b = new Student("A",numbers2);  
  
 List<Student> students = new ArrayList<>();  
 students.add(a);  
 students.add(b);  
  
 List<List<Integer>> collect = students.stream().map(student -> student.getPhoneNumbers()).collect(Collectors.*toList*());  
  
 System.*out*.println(collect);  
  
 List<Integer> collect2 = students.stream().flatMap(student -> student.getPhoneNumbers().stream()).collect(Collectors.*toList*());  
 System.*out*.println(collect2);  
 }  
  
  
}  
  
class Student {  
 private String clazz;  
 private List<Integer> phoneNumbers;  
  
 public Student(String clazz, List<Integer> phoneNumbers) {  
 this.clazz = clazz;  
 this.phoneNumbers = phoneNumbers;  
 }  
  
 public String getClazz() {  
 return clazz;  
 }  
  
 public void setClazz(String clazz) {  
 this.clazz = clazz;  
 }  
  
 public List<Integer> getPhoneNumbers() {  
 return this.phoneNumbers;  
 }  
  
 public void setPhoneNumbers(List<Integer> phoneNumbers) {  
 this.phoneNumbers = phoneNumbers;  
 }  
}

package stremz;  
  
import java.util.\*;  
import java.util.stream.Collectors;  
  
public class StreamMain {  
 public static void main(String[] args) {  
  
 List<Integer> numbers = Arrays.*asList*(2815,5678,2234);  
 Student a = new Student("A",numbers);  
  
 List<Integer> numbers2 = Arrays.*asList*(1111,2222,3333);  
 Student b = new Student("A",numbers2);  
  
 List<Student> students = new ArrayList<>();  
 students.add(a);  
 students.add(b);  
  
 List<List<Integer>> collect = students.stream().map(student -> student.getPhoneNumbers()).collect(Collectors.*toList*());  
  
 System.*out*.println(collect);  
  
 List<Integer> collect2 = students.stream().flatMap(student -> student.getPhoneNumbers().stream()).collect(Collectors.*toList*());  
 System.*out*.println(collect2);  
 }  
  
  
}  
  
class Student {  
 private String clazz;  
 private List<Integer> phoneNumbers;  
  
 public Student(String clazz, List<Integer> phoneNumbers) {  
 this.clazz = clazz;  
 this.phoneNumbers = phoneNumbers;  
 }  
  
 public String getClazz() {  
 return clazz;  
 }  
  
 public void setClazz(String clazz) {  
 this.clazz = clazz;  
 }  
  
 public List<Integer> getPhoneNumbers() {  
 return this.phoneNumbers;  
 }  
  
 public void setPhoneNumbers(List<Integer> phoneNumbers) {  
 this.phoneNumbers = phoneNumbers;  
 }  
}

Student A scored 56, 78, 98

Student B scored 67,22,35

Student C scored 99,1,92

Find the highest score

package stremz;  
  
import java.util.\*;  
import java.util.stream.Collectors;  
  
public class StreamMain {  
 public static void main(String[] args) {  
  
 List<Integer> numbers = Arrays.*asList*(56,78,98);  
 Student a = new Student("nischal",numbers);  
  
 List<Integer> numbers2 = Arrays.*asList*(67,22,35);  
 Student b = new Student("Bima",numbers2);  
  
 List<Integer> numbers3 = Arrays.*asList*(99,1,92);  
 Student c = new Student("Pharaya",numbers3);  
  
 List<Student> students = new ArrayList<>();  
 students.add(a);  
 students.add(b);  
 students.add(c);  
  
 List<Integer> collect = students.stream().flatMap(student -> student.getPhoneNumbers().stream()).collect(Collectors.*toList*());  
  
 System.*out*.println(collect);  
 int max = collect.get(0);  
 for (Integer a3 : collect) {  
 if (a3 > max) {  
 max = a3;  
 }  
 }  
  
 System.*out*.println(max);  
 }  
  
  
}  
  
class Student {  
 private String name;  
 private List<Integer> scores;  
  
 public Student(String clazz, List<Integer> phoneNumbers) {  
 this.name = clazz;  
 this.scores = phoneNumbers;  
 }  
  
 public String getClazz() {  
 return name;  
 }  
  
 public void setClazz(String clazz) {  
 this.name = clazz;  
 }  
  
 public List<Integer> getPhoneNumbers() {  
 return this.scores;  
 }  
  
 public void setPhoneNumbers(List<Integer> phoneNumbers) {  
 this.scores = phoneNumbers;  
 }  
}

package stremz;  
  
import java.util.\*;  
import java.util.stream.Collectors;  
  
public class StreamMain {  
 public static void main(String[] args) {  
  
 List<Integer> numbers = Arrays.*asList*(56,78,98);  
 Student a = new Student("nischal",numbers);  
  
 List<Integer> numbers2 = Arrays.*asList*(67,22,35);  
 Student b = new Student("Bima",numbers2);  
  
 List<Integer> numbers3 = Arrays.*asList*(99,1,92);  
 Student c = new Student("Pharaya",numbers3);  
  
 List<Student> students = new ArrayList<>();  
 students.add(a);  
 students.add(b);  
 students.add(c);  
  
 List<Integer> collect = students.stream().flatMap(student -> student.getPhoneNumbers().stream()).collect(Collectors.*toList*());  
  
 System.*out*.println(collect);  
 int max = collect.get(0);  
 for (Integer a3 : collect) {  
 if (a3 > max) {  
 max = a3;  
 }  
 }  
  
 System.*out*.println(max);  
 }  
  
  
}  
  
class Student {  
 private String name;  
 private List<Integer> scores;  
  
 public Student(String clazz, List<Integer> phoneNumbers) {  
 this.name = clazz;  
 this.scores = phoneNumbers;  
 }  
  
 public String getClazz() {  
 return name;  
 }  
  
 public void setClazz(String clazz) {  
 this.name = clazz;  
 }  
  
 public List<Integer> getPhoneNumbers() {  
 return this.scores;  
 }  
  
 public void setPhoneNumbers(List<Integer> phoneNumbers) {  
 this.scores = phoneNumbers;  
 }  
}