## **1 – Data type**

Để tiết kiệm bộ nhớ, Java 'tái sử dụng' 'reuses' tất cả các đối tượng wrapper có giá trị nằm trong các phạm vi sau:

All Boolean values (true and false)  
All Byte values  
All Character values from \u0000 to \u007f (i.e. 0 to 127 in decimal)  
All Short and Integer values from -128 to 127

Tuy nhiên, một điều đáng chú ý là khi bạn tạo một primitive wrapper bằng từ khóa mới, một đối tượng mới sẽ được tạo và một đối tượng được lưu trong bộ nhớ đệm, ngay cả khi có sẵn, sẽ không được sử dụng. Ví dụ:

Integer i = 10; //Wrapper được tạo mà không sử dụng từ khóa new và do đó được lưu vào bộ nhớ đệm.

Integer j = 10; // Đối tượng được lưu trong bộ nhớ cache được sử dụng lại. Không có đối tượng mới được tạo.  
Integer k = new Integer(10); // Đối tượng mới được tạo ra. Đối tượng được lưu trong bộ nhớ đệm không được sử dụng lại.  
=> i == j is true nhưng i == k is false.

Nếu Integer i = 128; => Đối tượng mới được tạo

|  |  |  |
| --- | --- | --- |
| |  | | --- | | "String".replace('g','g')=="String" |  |  | | --- | | replace returns the same object if there is no change. | |

- byte b = 128 ; => khai báo vợt giá trị sẽ không complite

StringBuilder sb = new StringBuilder();  
sb.ensureCapacity(100);

- private StringBuilder sb = new StringBuilder();

=> sb.delete(0, sb.length());’

- StringBuilder, String is a final class. => method length() of String class is a final method.

switch (Integer.*parseInt*(args[1])) {  
 case 0:  
 var b = false;  
 break;  
 case 1:  
 b = true;  
 break;  
}

=> hợp lệ

Integer i1 = 1;  
int i3 = 1;  
System.*out*.println(i1 == 13);

=> true

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | | int ia[][] = { {1, 2}, null }; | | |  |  | | --- | --- | | |  | | --- | | var ia = new int[][]{ {1, 2}, null }; | | | |

String s = "hell";  
StringBuilder sb = new StringBuilder("well");  
System.*out*.println(s + sb);

=> không lỗi

## **2 – Java OOA**

public class TestOuter {  
 public static class TestInner {  
 public void sayIt() {  
 System.*out*.println("hello");  
 }  
 }  
  
 public static void main(String[] args) {

new TestOuter.TestInner(;

new TestInner();   
 }  
  
 static void doElements(List l) {  
 l.add("string"); //1 System.out.println(l.get(0)); //2 }  
 }  
  
}

- constructor có thể throws exception.

- Mọi **emun** đều implements **Comparable**

**-** Nếu bạn có inner class ( non-static class) bạn có thể sử dụng trong non-static method của outer class

class Outer {  
 private void Outer() {  
 }  
  
 protected class Inner {  
 }  
}

=> Constructor for Inner is protected.

interface Eatable {  
 int *types* = 10;  
}  
  
class Food implements Eatable {  
 public static int *types* = 20;  
}  
  
class Fruit extends Food implements Eatable {  
 public static void main(String[] args) {  
 types = 30;  
 System.*out*.println(types);  
 }  
}

=> lỗi tại 2 và 3 vì types là biến ẩn

You cannot access a non-final static field from an enum's constructor.

Java's support for multiple inheritance of type.

interface I1{ void m1() throws IOException;}  
interface I2{ void m1() throws SQLException;}  
class TestClass implements I1, I2  
{  
 public void m1()  
 {  
 System.*out*.println("Hi there");  
 }  
}

- An enum cannot be defined inside any method or constructor.

## **3 – Controlling Program Flow**

for (var value : arr) {  
 if (counter >= 5) {  
 break;  
 } else {  
 continue;  
 }  
 if (value > 4) {  
 arr[counter] = value + 1;  
 }  
 counter++;  
}

=> Lỗi do từ if(value > 4) luôn luôn không được chạy.

'c' = 99  = 1100011  
'd' = 100 = 1100100  
'c'|'d'    =  1100111  <== 103, same as 'g'.

void test(Object a){}

void test(IOException a){}

void test(FileNotFoundException a){}

=> test(null) sẽ gọi function chi tiết nhất là: void test(FileNotFoundException a){}

- A boolean cannot be used for a switch statement.

switch (day) {  
 case *MONDAY*:  
 TUESDAY:  
 WEDNESDAY:  
 THURSDAY:  
 FRIDAY:  
 System.*out*.println("working");  
 case *SATURDAY*:  
 SUNDAY:  
 System.*out*.println("off");  
}

SUNDAY: (được xem là label)

if (args[0].equals("open"))   
 if (args[1].equals("someone")) System.*out*.println("Hello!");  
else System.*out*.println("Go away " + args[1]);

⬄

if (args[0].equals("open"))   
 if (args[1].equals("someone")) {  
 System.*out*.println("Hello!");  
 } else {  
 System.*out*.println("Go away " + args[1]);  
 }

while (false) { x=3; }

=> lỗi do while không thực hiện

## **4 - Arrays and Collections**

- No **Comparator** is supplied to the **sorted()** method. Therefore, at run time, when the sorted method tries to cast this class object to **Comparable**, it will fail.

**- java.util.List** và **java.util.Set** có **of** và **copyOf** để tạo list/set không thể sửa đổi.

List<Number> col = new ArrayList<>();  
col.add(1);  
var list1 = List.*of*(col);   
col.add(2);   
var list2 = List.*copyOf*(col);   
System.*out*.println(list1+", "+list2);

=> [[1, 2]], [1, 2]

var numA = new Integer[]{1, 2};  
var list1 = List.*of*(numA);  
numA[0] = 2;  
var list2 = List.*copyOf*(list1);  
System.*out*.println(list1 + " " + list2);

=> [1, 2] [1, 2] (khác bên trên)

- a [ (a = b)[3] ] ⬄ a[0]

- (twoD[0] == twoD.clone()[0] => true

- Queue có cấu trúc FIFO (First In First Out tức là thêm vào cuối và xóa khỏi phía trước), nó có các phương thức Offer(e)/add(e)(để thêm một phần tử vào cuối hoặc đuôi) và poll( )/remove()(để xóa một phần tử khỏi phía trước hoặc phần đầu) cho mục đích này.

Stack là cấu trúc LIFO (Last In First Out tức là thêm vào phía trước và xóa khỏi phía trước), nên nó cung cấp các phương thức push(e) và pop() cho mục đích này, trong đó push thêm vào phía trước và pop loại bỏ khỏi mặt trước.

Deque<Integer> d = new ArrayDeque<>();  
d.push(1);  
d.push(2);  
d.push(3);  
=> 3,2,1

Queue sẽ là 1,2,3

- mảng ko khai báo giá trị:

int[] b1 = new int[2];  
System.*out*.println(b1[0]);

// output: 0

=> nhận giá trị mặc định.

- return new ArrayList(); // sẽ loại bỏ các thông tin về type

- HashMap hỗ trợ thêm key null cũng như value null nhưng ConcurrentHashMap thì không

public class Counter<T> { /  
 T t;  
  
 public <T> int count(T[] ta, T t) {  
 this.t = t; // lỗi  
 int count = 0;  
 for (T x : ta) {  
   
 count = x == t ? count + 1 : count;  
 }  
 return count;  
 }  
}

=> biến generic khi khia báo **Counter<T>** khác với T khi khai báo method **public <T> int count(T[] ta, T t)**

- Arrays.compare(a, b); trả về số pahanf tử trênh lệch giữa hai mảng. Nếu cùng độ dài, so sánh từng phần tử trong mảng

interface Player<E>{ void play(E e); }

class Bowler implements **Player<Guitar>**{     public void play(Guitar o){ } }

- var[] sA = new String[1] ; sA[0] = "aaa";

var array2D = new int[][]  { { 0, 1, 2, 4}, {5, 6}};

=> var không cần khai báo kiểu, chỉ vần **var sA**

List<? super Dooby> dataList => những class cha của Dooby và Dooby

List<? extends Dooby> dataList => những class con Dooby và Dooby

System.out.print(twoD[0] == twoD.clone()[0]);

=> true

The add method of ArrayList returns a boolean.

public class PlaceHolder<K, V> {  
 private K k;  
 private V v;  
  
 public PlaceHolder(K k, V v) {  
 this.k = k;  
 this.v = v;  
 }  
  
 public K getK() {  
 return k;  
 }  
  
 public static <X> PlaceHolder<X, X> getDuplicateHolder(X x) {  
 return new PlaceHolder<X, X>(x, x);  
 }  
  
 public static void main(String[] args) {  
 PlaceHolder<String, String> ph1 = PlaceHolder.*getDuplicateHolder*("b");  
 PlaceHolder<String, String> ph2 = PlaceHolder < String >.*getDuplicateHolder*("b");  
 PlaceHolder<String, String> ph3 = PlaceHolder <>.  
 *getDuplicateHolder*("b");  
 PlaceHolder<> ph4 = new PlaceHolder<String, String>("a", "b");  
 PlaceHolder<?, ?> ph5 = new PlaceHolder(10, 10);  
 }  
}

=> getDuplicateHolder is a method and not a constructor.  
1,3,4 lỗi

|  |  |
| --- | --- |
| |  | | --- | | PlaceHolder<String, String> ph2 = PlaceHolder.<String>getDuplicateHolder("b"); | |

=> không lỗi

## **5 - Exception Handling**

-It improves code because error handling code is clearly separated from the main program logic.

It allows creation of new exceptions that are custom to a particular application domain.

- Throwable là lớp cao nhất trong hệ thống phân cấp ngoại lệ và hầu như không bao giờ được sử dụng trực tiếp.

- Trong try-with-resource ,bạn có thể sử dụng một biến được khai báo trước câu lệnh try trong khối try-with-resource miễn là nó là biến final or effectively final

- finally sẽ vẫn thực hiện khi catch throw ra exception

public static void main(String args[]) throws Exception {  
 try {  
 *m1*();  
 System.*out*.println("A");  
 }catch (Exception e){  
 throw new Exception("ok");  
 } finally {  
 System.*out*.println("B");  
 }  
 System.*out*.println("C");  
}  
  
public static void m1() throws Exception {  
 throw new Exception();  
}

- Throwable[] ta = e.getSuppressed();

=> It will not print any thing.

- Nếu một ngoại lệ được đưa ra trong khối try-with-resources thì đó là ngoại lệ mà người gọi nhận được. Nhưng trước khi khối thử trả về, phương thức close() của tài nguyên được gọi và nếu phương thức close() cũng đưa ra một ngoại lệ thì ngoại lệ này sẽ được thêm vào ngoại lệ ban đầu dưới dạng ngoại lệ bị chặn.

public static void main(String[] args) {  
 try(A a = new A();) {  
 throw new RuntimeException("b");  
 } catch (Exception e) {  
 System.*out*.println(e);  
 }  
 }  
  
  
  
class A implements AutoCloseable {  
 @Override  
 public void close() throws Exception {

System.*out*.println("ok");   
 throw new RuntimeException("Close");  
 }  
}

=> output: ok -> B

int x = 1;  
int y = 0;  
if (x / y) System.*out*.println("Good");  
else System.*out*.println("Bad");

=> lỗi do điều kiện if phải trả về boolean

} catch(IOException | IndexOutOfBoundsException e)  
  
{  
 e = new FileNotFoundException();  
 e.printStackTrace();  
}

=> The exception parameter in a multi-catch clause is implicitly final.( khia báo này sai)

## **6 - Concurrency**

- private final ReentrantLock lock = new ReentrantLock();

Đặt nó ở trạng static sẽ khiến tất cả đối tượng sử dụng cùng một khóa. Điều này có nghĩa là chỉ có thể sử dụng một đối tượng bất cứ lúc nào, điều này sẽ làm giảm đáng kể hiệu suất và không như mong muốn.

- Callable

public interface Callable<V>{   
 V call() throws Exception;  
}

abstract class Calculator {  
 abstract void calculate();  
  
 public static void main(String[] args) {  
 System.*out*.println("calculating");  
 Calculator x = null;  
 x.calculate();  
 }  
}

=> It will print calculating and then throw NullPointerException.

- AtomicInteger ai = new AtomicInteger(5);

int x = ai.incrementAndGet();

int x = ai.addAndGet(1);

public AtomicInteger status = new AtomicInteger(0);

|  |  |
| --- | --- |
| |  | | --- | | status.compareAndSet(oldstatus, newstatus); | |

## **7 - Java I/O API**

public void usePrintWriter(PrintWriter pw) {  
 boolean bval = true;  
 pw.println(bval);  
}

**public Path resolveSibling(String other)** or **public Path resolveSibling(Path other)**

- Giải quyết đường dẫn đã cho so với đường dẫn gốc của đường dẫn này.

static Path p1 = Paths.get("c:\\main\\project\\Starter.java");

String root = p1.getRoot().toString();

=> output: c:\

- **java.nio.file.NoSuchFileException**: test.txt

This exception will be thrown when the program tries to create a BufferedReader to read the file specified by the Path object.

- **Console.readPassword**("%s", "Enter Password :") => trả về 1 mảng String

- Stream<String> ref = new BufferedReader(new FileReader("c:\\temp\\test.txt")).lines();

- I/O operations (such as opening a stream on a file, reading or writing from/to a file) throw IOException

Console c = System.*console*();  
String line = c.readLine("Please enter your name:");  
System.*out*.println("Hello, " + line);

=> Line //2 will throw NullPointerException.

public static void main(String args[]) throws Exception {  
  
 Path p1 = Paths.*get*("c:\\temp\\test1.txt");  
 Path p2 = Paths.*get*("c:\\temp\\test2.txt");  
 *copy1*(p1, p2);  
 if (Files.isSameFile(p1, p2)) {  
 System.*out*.println("file copied");  
 } else {  
 System.*out*.println("unable to copy file");  
 }  
   
}  
  
public static void copy1(Path p1, Path p2) throws Exception {  
 Files.copy(p1, p2, StandardCopyOption.COPY\_ATTRIBUTES, StandardCopyOption.REPLACE\_EXISTING);  
}

=> Phương thức Files.copy sẽ sao chép file test1.txt vào test2.txt. Nếu test2.txt không tồn tại, nó sẽ được tạo. Tuy nhiên, phương thức Files.isSameFile không kiểm tra nội dung của tệp. Nó nhằm kiểm tra xem hai đối tượng đường dẫn có giải quyết được cùng một tệp hay không. Trong trường hợp này, chúng không phải như vậy và do đó, nó sẽ trả về sai.

var p1 = Paths.*get*("c:\\code\\java\\PathTest.java");  
System.*out*.println(p1.getName(3).toString());

=> It will throw IllegalArgumentException

## **8 - Module**

- javac --module-source-path src -d out --module foo.bar

- module-info.java must not be empty.

- module-info.java is compiled into module-info.class by the compiler.

- java -classpath graphs.jar com.xcomp.graphs.Main

- java --module-path mathutils.jar --module abc.math.utils/abc.utils.Main

## **9 - Java JDBC**

- Put the driver jar in the class path.

- **PreparedStatement** cho phép một số loại SQL bổ sung như BLOB và CLOB.

- ps.setNull(3, Types.INTEGER); ps.setString(4, null);

- requires java.desktop;

Connection con = DriverManager.getConnection(dbURL);

con.commit();

try (Connection c = ds.getConnection(); var stmt = c.prepareCall("{call insert\_name(?,?)}")) {  
// CallableStatement stmt = conn.prepareCall("{CALL my\_storedprocname(?, ?, ?)}");  
 stmt.setObject("surname", "Doe", JDBCType.*VARCHAR*);  
 stmt.setObject("GIVENNAME", "Jane", JDBCType.*VARCHAR*);  
 stmt.execute();  
 }

Connection c = ds.getConnection();  
Statement stmt = c.createStatement();  
String qr = "select \* from PERSON";  
try (ResultSet rs = stmt.executeQuery(qr); PreparedStatement ps1 = c.prepareStatement("update PERSON set taxid = 'NNNN' where id=?");) {  
 while (rs.next()) {  
 int id = rs.getInt(1);  
 ps1.setInt(1, id);  
 ps1.executeUpdate();  
 System.*out*.println("Updated " + rs.getString(3)); //2 } }  
 }  
}

=> in ra dữ liệu trước khi update

try (Statement stmt = connection.createStatement();   
 ResultSet rs = stmt.executeQuery("select \* from STUDENT");) {  
 while (rs.next()) { //do something with the row   
 }  
}

1. **Create a CallableStatement**  
   CallableStatement stmt = conn.prepareCall("{CALL my\_storedprocname(?, ?, ?)}");   
     
   (OR if the procedure returns a value (in which case, it is called "stored function"):  
     
   CallableStatement stmt = conn.prepareCall("{? = CALL my\_storedprocname(?, ?, ?)}");
2. **Set parameter values**  
   CallableStatement inherits from PreparedStatement, therefore, you can use various setXXX methods to set the input values  
     
   stmt.setString(1,"johndoe");  
   stmt.setObject(2, 20, java.sql.Types.INTEGER);//using the setObject method with the "type" argument   
     
   In addition to the above, a CallableStatement also allows you to set a parameter by name aka "parameter marker" like this:  
     
   stmt.setInt("NAME","johndoe");  
   //stmt.setInt("name","johnd");//parameter marker is case insensitive  
   stmt.setObject("DEPTID", 20, java.sql.Types.INTEGER);
3. **Register OUT (and INOUT) parameters**  
   To use the out parameter (assuming that the third parameter is an OUT parameter in this stored procedure), you must first "register" it like this:  
     
   stmt.registerOutParameter(3, java.sql.Types.INTEGER);  
   OR  
   stmt.registerOutParameter("AMOUNT", java.sql.Types.INTEGER);
4. **Retrieve the result**  
   The procedure can be executed and the result can be retrieved as follows:  
   boolean hasResults = stmt.execute(); //check if the stored proc has returned one or more ResultSets  
   while (hasResults) {  
      ResultSet rs = stmt.getResultSet();  
      // process result set  
      hasResults = stmt.getMoreResults();  
   }  
   int outvalue = stmt.getInt(3);//retrieve out value

## **10 – Streams và Lambda**

-Only non-terminal operations such as filter, peek, map, distinct, sorted, and skip are lazy. All terminal operations such as forEach, reduce, collect, count, min, max, allMatch, findAny are eager.

- Streams support several aggregate operations(một số hoạt động tổng hợp) such as forEach, count, average, and sum.

- reduction operations: reduce() and collect(), as well as multiple specialized reduction forms such as sum(), max(), or count().

- The Collector created by Collectors.toMap throws java.lang.IllegalStateException if an attempt is made to store a key that already exists in the Map.

- IntStream does not have parallelStream() method.

- java.util.List interface defines a default void sort(Comparator<? super E> c) method

=> yêu cầu Comparator

- Optional.of method sẽ throws NullPointerException nếu bạn cố gắng tạo **Optional** cógiá trị null with a null.

- public V merge(K key, V value, BiFunction<? super V,? super V,? extends V> remappingFunction)

=> Nếu key không tồn tại trong Map, sẽ thêm key,vlue mới

- Consumer x = (String msg)->{ System.out.println(msg); };

=> Lỗi vì Consumer x sẽ có đầu ra là Object (Consumer<String> thì sẽ đúng)

- new Random().doubles(10).forEach(System.out::print);

Random r = new Random();

DoubleStream rDoubles = r.generate().limit(10);

rDoubles.forEach(System.out::print);

- List s1 = new ArrayList( ); => s1.remove(obj) sẽ trả về boolean

IntStream is1 = IntStream.range(1, 3);

IntStream is2 = IntStream.rangeClosed(1, 3);

IntStream is3 = IntStream.concat(is1, is2);

Object val = is3.boxed().collect(Collectors.groupingBy(k->k)).get(3);

=> List<Integer>> containing: {1=[1, 1], 2=[2, 2], 3=[3]}

List<String> names = Arrays.*asList*("charles", "chuk", "cynthia", "cho", "cici");  
long x = names.stream().filter(name -> name.length() > 4).collect(Collectors.*counting*());  
System.*out*.println(x);

=> Collectors.counting()); đếm số lượng element trả về kiểu long.

UnaryOperator<Integer> func = x->x+10;

Local variables referenced from a lambda expression must be final or effectively final.

public class ArrayList<E> extends AbstractList<E>  
 implements List<E>, RandomAccess, Cloneable, java.io.Serializable

Employee e = new Employee(); //2

System.out.println(validateEmployee(e, e->e.age<10000)); //3

=> Lỗi tại 3

## **11 – Localization**

- Locale.setDefault(Locale.Category.FORMAT, Locale.US);

- Locale myLocale = Locale.getDefault();

- Locale myLocale = Locale.US;

- Locale myLocale = new Locale("ru", "RU");

- Locale loc = new Locale("fr", "FR");

DateTimeFormatter df = DateTimeFormatter.ofPattern("dd MMM yyyy", loc);  
System.out.println(df.format(d));

Format formatter =  NumberFormat.**getCurrencyInstance**(jp); // hoặc NumberFormat

DateFormat df = new DateFormat(DateFormat.LONG); // DateFormat là abstract class và java.text.DateFormat class cung cấp các static getXXXInstance methods

ResourceBundle rb = ResourceBundle.getBundle("test.MyBundle", locale);   
Object obj = rb.getObject("key1");  
String[] vals =  rb.getStringArray("key2");

## **12 - Annotation**

@Retention(value = RUNTIME)  
@Target(value = {TYPE\_USE, TYPE\_PARAMETER})  
public @interface NonNull {  
}

var str = (@NonNull String) "";

@NonNull String str = "";

static void doElements(List l) {  
 l.add("string"); //1 System.out.println(l.get(0)); //2  
}

Trình biên dịch tạo cảnh báo "unchecked or unsafe operations" khi nhận thấy khả năng bộ sưu tập bị hỏng do chèn các phần tử không tương thích.

public @interface Authors {  
 Author[] value();  
}  
  
@Repeatable(Authors.class)  
public @interface Author {  
 int id() default 0;  
  
 String value();  
}

=> **Author** có thể khai báo nhiều lần trong khi **Authors** thì không.

- You cannot set an annotation element (or its values, if it is an array) to null.

static void printElements(List<String>... la) {  
 for (List<String> l : la) {  
 System.*out*.println(l);  
 }  
}

=> Neither of the lines //1 and //2 will cause the compile to generate a warning.