# Тест

1. Неявное преобразование типов происходит в случае, если

* Оба типа совместимы
* Происходит преобразование числа с плавающей точкой в целое число
* Длина целевого типа меньше или равна длины исходного типа
* Длина целевого типа больше или равна длине исходного типа

1. Значение числа b будет равно

double i = 13.456;

int b = (int) i;

* 13
* 14
* 13.456
* 13.0

1. StringBuilder лучше использовать вместо StringBuffer в случае если

* Приложение многопоточное
* Приложение однопоточное
* Всегда
* Никогда

1. Метод matches(String pattern, CharSequence input), класса Pattern возвращает true если

### 

* Последовательность символов input полностью соответствует шаблону строки pattern
* Последовательность символов input включает в себя помимо других символов, шаблон строки pattern
* Последовательность символов input содержит в себе минимум 1 вхождение pattern
* Нет правильного ответа

1. Что будет выведено в результате работы метода

String input = "Functional testing is a quality assurance (QA) process[1] and a type of black-box testing that bases its test cases on the specifications of the software component under test. Functions are tested by feeding them input and examining the output, and internal program structure is rarely considered (unlike white-box testing)";

Pattern pattern = Pattern.compile("test(\\w\*)");

Matcher matcher = pattern.matcher(input);

while (matcher.find()) {

System.out.println(matcher.group());

}

* True
* False
* Слова, начинающиеся на test
* Все слова "test"

1. Что делает метод ordinal() в Enum

* Возвращает порядок, в котором экземпляры Enum обозначены внутри Enum
* Сортирует Enum от минимального значения к максимальному
* Сортирует Enum от максимального значения к минимальному
* Очищает Enum

1. Если с помощью log4j необходимо писать логируемые сообщения в файл, записывая новый файл по достижении определенного объема, то используется аппендер

* FileAppender
* DailyRollingFileAppender
* RollingFileAppender
* SMTPAppender

1. Что описывает интерфейс Map в библиотеке коллекций Java(Java Collections Framework)

* Карту всех возможных методов коллекций
* Коллекцию, состоящую из пар "ключ — значение"
* Расширяет возможности работы с интерфейсом Collection
* Интерфейс позволяет в процессе работы получить все переменные, объявленные в программе на данный момент

1. Ключевое отличие интерфейсов List и Set

* Set не может содержать в себе повторяющиеся элементы
* Set содержит в себе пары ключ-значение
* Set не поддерживает расширение коллекции
* Отличий нет

1. Java Reflection позволяет

* Определить класс объекта.
* Получить информацию о модификаторах класса, полях, методах, конструкторах и суперклассах.
* Выяснить, какие константы и методы принадлежат интерфейсу.
* Создать экземпляр класса, имя которого неизвестно до момента выполнения программы.
* Получить и установить значение свойства объекта.
* Вызвать метод объекта по имени.
* Создать новый массив, размер и тип компонентов которого неизвестны до момента выполнения программ.

1. В результате каких команд на консоль будет выведено "22"?

* System.out.println("2" + 1 + 1);
* System.out.println(1 + "1" + "2");
* System.out.println("2" + "2");
* System.out.println(1 + 1 + "2");

1. Что будет результатом компиляции и выполнения данного кода:

public class Main {

public static void main(String[] args) {

StringBuilder strBuffer = new StringBuilder("Java");

char c = strBuffer.charAt(0);

strBuffer.setCharAt(0, 'c');

System.out.println(c);

System.out.println(strBuffer);

}

}

* J

cava

* Ошибка компиляции
* J

Java

* ava

J

1. Что будет результатом компиляции и выполнения данного кода:

public class Main {

public static void main(String[] args) {

StringBuffer strBuffer = new StringBuffer("assembler");

strBuffer.delete(0,2);

System.out.println(strBuffer);

strBuffer.deleteCharAt(6);

System.out.println(strBuffer));

}

}

* null
* Ошибка компиляции
* sembler

semble –при удалении лишней скобки в строке вывода strBuffer

* sembler

assember

1. Что будет результатом компиляции и выполнения данного кода:

public class Main {

public static void main(String[] args) {

String REGEX = "dog";

String INPUT = "The dog says meow. "+"All dogs say meow";

String REPLACE = "cat";

Pattern p = Pattern.compile(REGEX);

Matcher m = p.matcher(INPUT);

INPUT = m.replaceAll(REPLACE);

System.out.println(INPUT);

}

}

* cat
* Ошибка компиляции
* The cat says meow. All cats say meow
* The dog says meow. All dog say meow.

1. Что будет результатом компиляции и выполнения данного кода:

public enum Directions{

EAST,

WEST,

NORTH,

SOUTH

}

public static void main(String args[]) {

Directions dir = Directions.NORTH;

if(dir == Directions.EAST) {

System.out.println("Direction: East");

} else if(dir == Directions.WEST) {

System.out.println("Direction: West");

} else if(dir == Directions.NORTH) {

System.out.println("Direction: North");

} else {

System.out.println("Direction: South");

}

}

* Direction: South
* Direction: West
* Directions.NORTH
* Direction: East

1. Что будет результатом компиляции и выполнения данного кода:

public static void main(String args[]) {

HashSet<String> hset = new HashSet<String>();

// Adding elements to the HashSet

hset.add("Grapes");

hset.add("Mango");

hset.add("Apple");

hset.add("Orange");

hset.add("Fig");

hset.add("Apple");

hset.add("Mango");

}

* В коллекции hset 7 элементов
* Ошибка компиляции
* В коллекции hset 5 элементов
* Ошибка выполнения

1. Что будет результатом компиляции и выполнения данного кода:

public static void main(String args[]) {

TreeSet<Integer> tset = new TreeSet<Integer>();

tset.add(88);

tset.add(7);

tset.add(101);

tset.add(0);

tset.add(3);

tset.add(222);

System.out.println(tset);

}

* [88, 7, 101, 0, 3, 222]
* Ошибка компиляции
* [0, 3, 7, 88, 101, 222]
* [222, 101, 88, 7, 3, 0]

1. Что будет результатом компиляции и выполнения данного кода:

public static void main(String args[]) {

PriorityQueue<String> queue1 = new PriorityQueue<>();

queue1.offer("Oklahoma");

queue1.offer("Indiana");

queue1.offer("Georgia");

queue1.offer("Texas");

while (queue1.size() > 0) {

System.out.print(queue1.poll() + " ");

}

}

* Oklahoma Indiana Georgia Texas
* [Oklahoma Indiana Georgia Texas ]
* Georgia Indiana Oklahoma Texas
* [Georgia Indiana Oklahoma Texas ]

1. Что будет результатом компиляции и выполнения данного кода:

public static void main(String args[]) {

HashMap<Integer, String> hmap =

new HashMap<Integer, String>();

hmap.put(12, "12");

hmap.put(2, "2");

hmap.put(12, "1212");

hmap.put(49, "49");

hmap.put(3, "3");

for (Object aSet : hmap.entrySet()) {

Map.Entry mEntry = (Map.Entry) aSet;

System.out.println("key is: " + mEntry.getKey() +

" & Value is: " + mEntry.getValue());

}

}

* Ошибка выполнения
* Ошибка компиляции
* key is: 49 & Value is: 49

key is: 2 & Value is: 2

key is: 3 & Value is: 3

key is: 12 & Value is: 1212

* key is: 49 & Value is: 49

key is: 12 & Value is: 12

key is: 2 & Value is: 2

key is: 3 & Value is: 3

key is: 12 & Value is: 12

1. Что будет результатом компиляции и выполнения данного кода:

public static void main(String args[]) {

TreeMap<Integer, String> tmap = new TreeMap<>();

tmap.put(1, "Data1");

tmap.put(23, "Data2");

tmap.put(70, "Data3");

tmap.put(4, "Data4");

tmap.put(2, "Data5");

Set set = tmap.entrySet();

for (Object aSet : set) {

Map.Entry mentry = (Map.Entry) aSet;

System.out.print("key is: " + mentry.getKey() +

" & Value is: " + mentry.getValue());

}

}

* Ошибка выполнения
* Ошибка компиляции
* key is: 1 & Value is: Data1

key is: 2 & Value is: Data5

key is: 4 & Value is: Data4

key is: 23 & Value is: Data2

key is: 70 & Value is: Data3

* key is: 1 & Value is: Data1

key is: 23 & Value is: Data2

key is: 70 & Value is: Data3

key is: 4 & Value is: Data4

key is: 2 & Value is: Data5

1. Что будет результатом компиляции и выполнения данного кода:

static class Tuple<K, V> {

private final K k;

private final V v;

public Tuple(K key, V value) {

k = key;

v = value;

}

public String toString() {

return "KEY: " + k + ", VALUE: " + v;

}

}

public static void main(String[] args) {

Tuple<String, Integer> t = new Tuple<>("foo", 1);

Tuple<Integer, Integer> t2 = new Tuple<>(2, 1);

System.out.println(t);

System.out.println(t2);

}

* Ошибка компиляции
* Ошибка выполнения
* KEY: foo, VALUE: 1

KEY: 2, VALUE: 1

* KEY: [foo, 2], VALUE: [1, 1]

1. Что будет результатом компиляции и выполнения данного кода:

static class WithPrivateFinalField {

private int i = 1;

private final String s = "String S";

private String s2 = "String S2";

public String toString() {

return "i = " + i + ", " + s + ", " + s2;

}

}

public static void main(String[] args) throws Exception {

WithPrivateFinalField pf = new WithPrivateFinalField();

System.out.println(pf);

Field f = pf.getClass().getDeclaredField("i");

f.setAccessible(true);

f.setInt(pf, 47);

f = pf.getClass().getDeclaredField("s");

f.setAccessible(true);

f.set(pf, "MODIFY S");

f = pf.getClass().getDeclaredField("s2");

f.setAccessible(true);

f.set(pf, "MODIFY S2");

System.out.println(pf);

}

* Ошибка выполнения- так как не обработаны исключения
* i = 1, String S, String S2

i = 1, String S, String S2

* i = 1, String S, String S2

i = 47, String S, MODIFY S2-если исключения обработаны

* i = 1, String S, String S2

i = 47, MODIFY S, MODIFY S2