Práctica de Java: Procesador, memoria, pila de execución

Nome e apelidos:

SECCIÓN 1: PROCESADOR

No teu ordenador, escribe o seguinte programa nun ficheiro Main.java:

```java
public class Main {
    public static void main(String[] args) {
        while (true) {
            System.out.println("Ejecutando bucle infinito...");
        }
    }
}
```

Compílao e **executao**.

1.1. Se estás en Windows, pulsa **CTRL+ALT+SUPR** e abre o administrador de tarefas. Se estás en Linux, executa **top** nun terminal. Canta CPU (en porcentaxe %) está consumindo o proceso de execución do teu programa Java? Ten sentido? Por que?

Agora **detén** o proceso, que está a imprimir por consola de xeito infinito.

1.2. Como fixeches para deter o proceso? Que outras alternativas hai e que implicacións teñen?

🧠 Podes preguntarlle á IA. Recomendablemente, proba cada unha das formas. A aprendizaxe é algo activo (facer), non pasivo (ler)!

En diversos contextos de informática, unha “espera activa” ou “*polling*” consiste nun bucle que, indefinidamente, estáse a executar e consumir activamente CPU para obter información necesaria.

Pola contra, unha “espera pasiva” consiste na espera sen consumir CPU dun proceso a que outro proceso ou elemento do sistema lle mande unha notificación.

1.3. Sen preguntarlle á IA nin buscar en Internet… Cal das dúas cres que é a técnica máis eficiente?

1.4. Máis en detalle, que vantaxes e desvantaxes cres que ten cada unha? En que contextos cres que se poden aplicar?

⛔🧠 Non lle preguntes á IA. Escribe a túa resposta en base ao que creas. Despois, podes buscar a información pola túa banda, pero deixa neste PDF de práctica escrita a túa resposta orixinal.

SECCIÓN 2: ENTRADA E SAÍDA (E/S)

Reescribe o programa Main.java para que agora sexa este:

```java
import java.io.FileOutputStream;
import java.io.IOException;

public class Main {
    public static void main(String[] args) {
        byte[] data = new byte[1024 * 1024]; // 1 MB

        while (true) {
            try (FileOutputStream fos = new FileOutputStream("salida.bin", true)) {
                fos.write(data);
                fos.flush();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

2.1. Que operacións cres que efectúa este programa? Que “fai”?

🧠 Usa a túa intuición! Se despois de pensar, non o tes claro… Podes preguntarlle á IA.

Compílao e **executao**.

2.2. Se estás en Windows, pulsa **`CTRL+ALT+SUPR`** e abre o administrador de tarefas. Se estás en Linux, executa **`iotop`** nun terminal. O programa accede a disco constantemente, verdade? Cal é o ancho de banda (en MB/s) aproximado de E/S[1] que está a consumir?

Non esquezas **deter** o proceso e **borrar** o ficheiro **`salida.bin`**.

SECCIÓN 3: MEMORIA

3.1. Canto ocupa na memoria cada un dos seguintes tipos de variable en Java?

- int
- long
- float
- double
- char
- boolean

3.2. Nun programa Java un obxecto que garda dous números (int) ocupa espazo en memoria equivalente a 2 ints? Por que?

🧠 Esta parece unha pregunta na que a IA pode orientarche un pouquiño.

Se temos unha clase así:

```
public class Persoa {
  private String nome;
  private String apelidos;
  private int idade;
  private boolean estaMotivado;
}
```

3.3. Cando podes estimar que ocupará en memoria un obxecto desa clase?

1 Entrada/saída. Normalmente refírese a operacións de lectura ou escritura nun disco duro ou outro dispositivo

Agora, temos que diferenciar entre un obxecto (instancia dunha clase) e unha variable que referencia a ese obxecto.

Se o noso programa principal é así:

```
public class Main {
  public static void main(String [] args) {
    Persoa persoa1 = new Persoa();
  }
}
```

3.4. Que está ocupando espazo en memoria?

- ⦿ A instancia do obxecto tipo Persoa
- ○ A variable (referencia) persoa1 e o obxecto referenciado, é dicir, a instancia
- ○ A variable (referencia) persoa1

Se o programa fose así:

```
public class Main {
  public static void main(String [] args) {
    Persoa persoa1 = new Persoa();
    Persoa persoa2 = new Persoa();
    Persoa persoa3 = new Persoa();
  }
}
```

3.5. Cantas instancias e referencias habería consumindo memoria?

E se o programa fose así...

```
public class Main {
  public static void main(String [] args) {
    Persoa persoa1 = new Persoa();
    Persoa persoa2 = persoa1;
    Persoa persoa3 = persoa2;
  }
}
```

3.6. Cantas instancias e referencias haberá consumindo memoria?

Reescribe o ficheiro Main.java e pon este programa, que reserva memoria de xeito infinito.

```java
import java.util.ArrayList;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        List<byte[]> memoria = new ArrayList<>();

        while (true) {
            // Reserva 1 MB en cada iteración
            memoria.add(new byte[1024 * 1024]);

            System.out.println("Memoria reservada: " + memoria.size() + " MB");

            // Pequeña pausa para que puedas observarlo mejor
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Compílao e **execútao**.

Explora nun administrador de tarefas (**CTRL+ALT+SUPR**) en Windows ou coa orde **top** en Linux canta memoria está a consumir o proceso mentres se executa.

3.7. Canta memoria conseguiu reservar antes de “romper”? Canto tempo tardou máis ou menos?

3.8. Copia e pega aquí o erro que deu ao final da execución.

3.9. Agora executa de novo o programa, pero empregando **`java -Xmx64m Main`**. Cantos MB reservou antes de romperse a execución? Que é a opción -Xmx?

🧠 Como poderías averiguar sobre -Xmx sen usar a IA? Quizais despois de probar java -help no terminal… Chámate a atención a opción java -X?

3.10. Que é o “heap space”?

🧠 Emprega a IA ou busca en Internet se o precisas.

SECCIÓN 4: RECOLECTOR DE BASURA

4.1. Cando se crea un obxecto en Java, resérvase memoria, como xa vimos. Pero, cando volve a estar esta memoria dispoñible? Que é o recolector de basura en Java?

🧠 Emprega a IA ou busca en Internet se o precisas.

Vamos a facer unha pequena proba para ver o funcionamento do recolector de basura (Garbage Collector).

Abre un novo ficheiro PruebaGC.java e escribe o seguinte programa:

```java
public class PruebaGC {
    public static void main(String[] args) {
        for (int i = 0; i < 100000; i++) {
            byte[] basura = new byte[1024 * 1024]; // 1MB
        }
    }
}
```

Este programa é similar ao do exercicio 3.7, pero ten diferencias: (1) Aquí non hai unha pausa (Thread.sleep) e tampouco faise System.out.println. Tamén, (2) úsase un bucle for en lugar de un while.

4.2. Pero, cal é a diferencia máis importante con respecto ao uso da memoria RAM?

⛔🧠 Non uses IA. Usa a túa intuición e deixa escrita a túa resposta orixinal. Máis adiante vamos a facer unha práctica e comprobar cal é a solución.

Se ao executar java indicamos "-Xlog:gc" estamos a solicitar que se mostren logs do recolector de basura, e así vemos cómo a JVM libera a memoria non utilizada.

4.3. Agora, executa o programa con `java -Xmx64m -Xlog:gc PruebaGC.` Despois dun intre, **detén** o programa e pega un fragmento da saída producida.

4.4. Que significa un trozo da saída coma o seguinte: "**`31M->1M(64M)`**"?

🧠 Emprega a IA só se non te fías da túa intuición...

Executa agora o programa Main.java da sección 3 empregando logs do recolector de basura.

4.5. Ves a mesma cantidade de logs do recolector de basura?

- ◉ Sí, son os mesmos logs. Idénticos
- ○ Os logs parécense, pero non suceden coa mesma frecuencia
- ○ Os logs saen coa mesma frecuencia, pero cambia o tipo de recolector (por exemplo, Pause Young)

4.6. Cales son as túas conclusións? Como están a funcionar cada un dos programas? Cómo actúa o recolector de basura para cada un deles?

SECCIÓN 5: PILA DE EXECUCIÓN

O recolector de basura non é o único elemento que libera a memoria usada polos programas.

Cando falamos de invocar unha función ou procedemento, estamos a dicir que o contexto da execución cambia.

5.1. Que é a pila de execución e cómo funciona cando se invoca unha función ou procedemento?

🧠 É aconsellable apoiarte na IA para esta pregunta.

5.2. A pila de execución é algo exclusivo de Java?

🧠 A túa intuición coincidiu co que che dixo a IA?

SECCIÓN 6: EXTRA (NON AVALIABLE)

6.1. Que é unha List? Que implementacións de “List” hai e que características teñen?

6.2. Na linguaxe JavaScript, as variables declaradas con “var” son un tanto especiais. Por que? Debido a isto, úsanse con frecuencia?

6.3. Nun programa escrito en C, como se reserva memoria?

6.4. Nun programa escrito en C… Existe o recolector de basura? Por que? Como se libera a memoria reservada?

6.5. Se estamos a escribir programas que funcionan nunha máquina servidora… É importante ter en conta que a memoria se use da forma máis óptima posible? En que casos non é tan importante?