(A) CONOSCENZA TERMINOLOGICA

Dare una breve descrizione dei termini introdotti:

- **Classe Runtime**
- **getRuntime()**
- **exec()**
- **classe Process**
- **classe System**
- **exitValue()**
- **waitFor()**
- **stdin, stdout, stderror**
- **getProperty()**
- **destroy()**

(B) CONOSCENZA E COMPETENZA

Rispondere alle seguenti domande producendo anche qualche esempio

1) Conoscenza

1. Che *relazione esiste* tra la classe **Runtime** e il sistema operativo?
2. Quali *operazioni* sono possibili con un oggetto di classe **Runtime**?
3. A cosa serve il *valore di ritorno* di un'applicazione?
4. Che differenza c'è tra i metodi **waitFor**() ed **exitValue**()?
5. Cosa indicano **stdin**, **stdout** ed **stderr**?

2) Competenza

1. Come può essere prodotto il *valore di ritorno* di un'applicazione?
2. Qual è la sintassi *per creare un oggetto* di classe **Runtime**?
3. Qual è il *prototipo* dei due metodi **exec**() più comuni?
4. In quale modo è possibile cronometrare un'applicazione?

(C) ESERCIZI DI COMPRENSIONE

1. Per richiamare l'esecuzione di comandi del sistema operativo o di applicazioni varie in Java, si usa la classe …….
2. Ad ogni applicazione Java è associato un oggetto di classe …….. che, quindi non necessita di essere creato: è sufficiente l'istruzione ……..
3. Su un oggetto *comando*.di classe **Runtime** è possibile eseguire un comando o un'applicazione tramite il metodo ……., che crea un sottoprocesso.
4. Il valore di ritorno di un'applicazione lanciata con **exec**() da un programma può essere ottenuto mediante il metodo **waitFor**(), quando il programma termina, ma l'applicazione resta in esecuzione. Se l'applicazione lanciata termina (regolarmente o forzatamente) è possibile avere il valore di ritorno anche mediante **exitValue**().
5. Scrivere la sintassi e l'effetto dei due metodi ex**ec**(…) della classe **Runtime**:

Sintassi	Effetto

6. Scrivere l'effetto dei seguenti comandi della classe **Runtime**:

Sintassi	Effetto
static Runtime getRuntime()	
int availableProcessors()	
long freeMemory()	
long totalMemory()	

7. Nei sistemi operativi Unix e Unix-like, negli ambienti d'esecuzione del linguaggio C, C++ e Java, esistono dei dispositivi logici detti …… standard relativi all'…….., all'……. ed agli …… I loro nomi tipici sono, rispettivamente: …….., …… e ……

8. Scrivere l'effetto dei seguenti comandi della classe **Process**:

Sintassi	Effetto
abstract int exitValue()	
abstract int waitFor()	
abstract InputStream getInputStream()	
abstract OutputStream getOutputStream()	
abstract InputStream getErrorStream()	
abstract void destroy()	

9. Esaminare il seguente codice e descrivere l'output prodotto.

```
public class ExecCommand
{
   public static void main(String args[])
   { try
     {   Runtime rt = Runtime.getRuntime();
         Process proc = rt.exec("javac");
         int exitVal = proc.waitFor();
         System.out.println("Process exitValue: " + exitVal);
     }
     catch (Throwable t)
     {   t.printStackTrace();    }
   }
}
```

10. Esaminare il seguente codice di cattura dell'output, completare le parti mancanti e descriverne il funzionamento.

```
public static void getOut (Process p)
{try
  {InputStream stderr = p.getErrorStream();
   ......................... isr = new InputStreamReader (.......);
   BufferedReader br = new ........................ (isr);
   ............ line = null;
   System.out.println("<ERROR>");
   while ((line = br.readLine()) != .........)  System.out.println (line);
   System.out.println("</ERROR>");
  }
 catch (Exception e)  {  System.out.println("Errore");    }
}
```

(D) ESERCIZI DI APPLICAZIONE

1. Scrivere la procedura corretta *getOut*() che cattura lo stream di errore di un sottoprocesso.
2. Scrivere la procedura corretta *getOut*() che cattura lo stream di output di un sottoprocesso.
3. Creare un'applicazione Java che, servendosi della classe **Runtime**, mostri all'utente in modalità grafica, un menu comandi dal quale possa lanciare un programma di:
   a. videoscrittura;
   b. di gestione di fogli elettronici;
   c. gestione basi di dati;
   d. gestione testi, senza formattazione;
   e. prompt del sistema operativo.

   L'applicazione deve gestire le caselle di controllo della finestra, i pulsanti relativi a ciascuna delle operazioni richieste ed un pulsante di chiusura.
4. Creare un'applicazione Java che, servendosi della classe **Runtime**, mostri all'utente in modalità grafica, un menu dei seguenti comandi del sistema operativo:
   a. impostazione data e ora;
   b. check di una unità a dischi selezionabile;
   c. visualizzazione della versione del sistema operativo;
   d. impostazione del prompt;
   e. prompt del sistema operativo.

   L'applicazione deve gestire le caselle di controllo della finestra, i pulsanti relativi a ciascuna delle operazioni richieste ed un pulsante di chiusura.

(E) ESERCITAZIONI PRATICHE
Esercitazione n. 1

Obiettivi: utilizzo del metodo **System.currentTimeMillis( )**, variabili **long**
Problema: scrivere un'applicazione che carichi in modo casuale un vettore di interi. L'applicazione deve cronometrare l'operazione di caricamento, memorizzando l'istante di inizio e quello di fine dell'operazione e stampando la differenza. Eseguire le prove con dimensioni variabili del vettore da 100 a 1000, con incrementi di 100.

1. Scrivere l'algoritmo relativo alla applicazione richiesta:

Algoritmo …….
Inizio

Fine

2. Scrivere la codifica Java:

```
class …….
{
   public static void main(String args[])
   {

   }
}
```

Fine

3. Completare la tabella con i valori in uscita dal programma:
4. Considerazioni personali:

Dimensione vettore	Tempo (msec)

(E) ESERCITAZIONI PRATICHE
Esercitazione n. 2

Obiettivi: calcolo sperimentale della complessità e confronto con il cronometraggio mediante utilizzo **System.currentTimeMillis()**, variabili **long**
Problema: si consideri il problema della ricerca sequenziale di un elemento all'interno di un vettore di numeri interi.

Per realizzare l'esperienza, occorre:

1. Scrivere un'applicazione *Ricerca.java* che contenga la funzione:

   void RicercaSequenziale (**int** v[ ], **int** n, **int** x)

   che nel vettore *v[ ]* di dimensione *n* cerchi un elemento di valore *x*.
   Prevedere nell'applicazione, una variabile globale intera *nCicli* che riporti il numero di cicli eseguiti all'interno della funzione. Compilare e collaudare l'applicazione con un vettore caricato con valori interi progressivi di dimensione variabile e riportare i risultati nella tabella seguente:

Ricerca sequenziale (valutazione complessità)			
N	Valore di *nCicli*		Caso medio (calcolato)
	Caso ottimo x = 0	Caso pessimo x = N	
10			
50			
100			
500			
1000			
5000			
10000			

2. Modificare la funzione *RicercaSequenziale* (…) in modo da cronometrare il blocco di codice che svolge la ricerca. Pertanto, occorre eliminare il contatore *nCicli* e introdurre due variabili **long** chiamate *start* ed *end* nelle quali registrare il tempo iniziale e finale, mediante il metodo **System.currentTimeMillis()**, come descritto nelle lezioni. Introdurre una variabile globale **long** chiamata *tempo* a cui assegnare la differenza fra *end* e *start*. Come fatto nella prima parte dell'esperienza, compilare e collaudare l'applicazione con un vettore caricato con valori interi progressivi di dimensione variabile e riportare i risultati nella tabella seguente:

Ricerca sequenziale (valutazione complessità)			
N	Cronometraggio (*msec*)		Caso medio (calcolato)
	Caso ottimo x = 0	Caso pessimo x = N	
10			
50			
100			
500			
1000			
5000			
10000			

(E) ESERCITAZIONI PRATICHE
Esercitazione n. 3

Obiettivi: calcolo sperimentale della complessità e confronto con il cronometraggio mediante utilizzo del metodo **System.currentTimeMillis()**, variabili **long**
Problema: si consideri il problema della ricerca binaria di un elemento all'interno di un vettore di numeri interi.

Per realizzare l'esperienza, occorre:

1. Scrivere un'applicazione *Ricerca.java* che contenga la funzione:

   void RicercaBinaria (**int** v[ ], **int** n, **int** x)

   che nel vettore *v[ ]* di dimensione *n* cerchi un elemento di valore *x*.
   Prevedere nell'applicazione, una variabile globale intera *nCicli* che riporti il numero di cicli eseguiti all'interno della funzione. Compilare e collaudare l'applicazione con un vettore caricato con valori interi progressivi di dimensione variabile e riportare i risultati nella tabella seguente:

Ricerca binaria (valutazione complessità)			
N	Valore di *nCicli*		**Caso medio** (calcolato)
	Caso ottimo x = 0	Caso pessimo x = N	
10			
50			
100			
500			
1000			
5000			
10000			

2. Modificare la funzione *RicercaBinaria* (…) in modo da cronometrare il blocco di codice che svolge la ricerca. Pertanto, occorre eliminare il contatore *nCicli* e introdurre due variabili **long** chiamate *start* ed *end* nelle quali registrare il tempo iniziale e finale, mediante il metodo **System.currentTimeMillis()**, come descritto nelle lezioni. Introdurre una variabile globale **long** chiamata *tempo* a cui assegnare la differenza fra *end* e *start*. Come fatto nella prima parte dell'esperienza, compilare e collaudare l'applicazione con un vettore caricato con valori interi progressivi di dimensione variabile e riportare i risultati nella tabella seguente:

Ricerca binaria (valutazione complessità)			
N	Cronometraggio (*msec*)		Caso medio (calcolato)
	Caso ottimo x = 0	Caso pessimo x = N	
10			
50			
100			
500			
1000			
5000			
10000			