Soluzione esercizio 1

Rifattorizzazione in classe indipendente

Per prima cosa, va creata una classe, ad esempio *S1Esercizio1Thread,* che estenda la classe padre *Thread* e che contenga il metodo *run* della versione originale. Nel metodo main (che viene eseguito dal main thread) va sostituita la creazione dell'anonymous inner class con l'istanziazione della nuova classe S1Esercizio1Thread. Per poter far partire i thread creati è necessario invocare il metodo start per ogni istanza di S1Esercizio1Thread. A questo punto l'anonymous inner class è stata rifattorizzata in una classe indipendente.

```
class S1Esercizio1Thread extends Thread {
  @Override
  public void run() {
    long fibo1 = 1, fibo2 = 1, fibonacci = 1;
    for (int i = 3; i <= 700; i++) {
      fibonacci = fibo1 + fibo2;
      fibo1 = fibo2;
      fibo2 = fibonacci;
    }
    System.out.println(this + ": " + fibonacci);
  }
}

public class S1Esercizio1 {
  public static void main(String[] args) {
    Collection<Thread> allThreads = new ArrayList<Thread>();
    for (int i = 1; i <= 5; i++) {
      Thread t = new S1Esercizio1Thread();
      System.out.println("Main: creo " + t);
      allThreads.add(t);
    }

    for (Thread t : allThreads)
      t.start();

    for (Thread t : allThreads) {
      try {
        t.join();
        System.out.println("Main: " + t + " ha terminato");
      } catch (InterruptedException e) {
        /* Unhandled exception */
      }
    }
  }
}
```

Rifattorizzazione in Runnable

Per trasformare la classe in un *Runnable* è necessario sostituire *extends Thread* con *implements Runnable*. Nel metodo main invece di creare direttamente i thread con la logica d'esecuzione vanno creati i runnables. Ogni runnable va poi assegnato ad un thread (al momento della creazione), che si occuperà di eseguirlo.

```
class S1Esercizio1Runner implements Runnable {
  @Override
  public void run() {
    long fibo1 = 1, fibo2 = 1, fibonacci = 1;
    for (int i = 3; i <= 700; i++) {
      fibonacci = fibo1 + fibo2;
      fibo1 = fibo2;
      fibo2 = fibonacci;
    }
    System.out.println(this + ": " + fibonacci);
  }
}

public class S1Esercizio1 {
  public static void main(String[] args) {
    Collection<Thread> allThreads = new ArrayList<Thread>();
    for (int i = 1; i <= 5; i++) {
      Thread t = new Thread(new S1Esercizio1Runner());
```

```
      System.out.println("Main: creo " + t);
      allThreads.add(t);
    }

    for (Thread t : allThreads)
      t.start();

    for (Thread t : allThreads) {
      try {
        t.join();
        System.out.println("Main: " + t + " ha terminato");
      } catch (InterruptedException e) {
         /* Unhandled exception */
      }
    }
  }
}
```

Lambda expression

Per introdurre la lambda expression si sfrutta la Functional Interface *Runnable* che impone il metodo *run*. Riprendendo la soluzione precedente, quando viene creata una nuova Thread, invece d'istanziare una nuova classe *S1Esercizion1Runnable*, si specifica direttamente tramite una lambda expression l'implementazione del metodo run.

```
public class S1Esercizio1 {
  public static void main(String[] args) {
    Collection<Thread> allThreads = new ArrayList<Thread>();
    for (int ii = 1; ii <= 5; ii++) {
      int id = ii;
      /* Crea una nuova thread che eseguira' il runnable creato tramite lambda expression */
      Thread t = new Thread(() -> {
        long fibo1 = 1, fibo2 = 1, fibonacci = 1;
        for (int i = 3; i <= 700; i++) {
          fibonacci = fibo1 + fibo2;
          fibo1 = fibo2;
          fibo2 = fibonacci;
        }
        System.out.println("Lambda" + id + ": " + fibonacci);
      });
      System.out.println("Main: creata Lambda" + id + " e assegnata a " + t);
      allThreads.add(t);
    }
    for (Thread t : allThreads)
      t.start();

    for (Thread t : allThreads) {
      try {
        t.join();
        System.out.println("Main: " + t + " ha terminato");
      } catch (InterruptedException e) {
         /* Unhandled exception */
      }
    }
  }
}
```

Soluzione esercizio 2

```
import java.util.Random;

class S1Es2Worker implements Runnable {
  private int id;
  private long time;

  public S1Es2Worker(int id, long time) {
    this.id = id;
    this.time = time;
  }

  @Override
  public void run() {
    try {
      Thread.sleep(time);
    } catch (InterruptedException e) {
      /* Unhandled exception */
    }
    System.out.println("Thread " + id + " risveglio dopo " + time + " ms");
  }
}

public class S1Esercizio2 {
  public static void main(String[] args) {
    Random rand = new Random();
    // Create runnables
    Thread t1 = new Thread(new S1Es2Worker(0, 1500 + rand.nextInt(501)));
    Thread t2 = new Thread(new S1Es2Worker(1, 1500 + rand.nextInt(501)));

    System.out.println("Partono tutti i threads.");
    // Start threads
    t1.start();
    t2.start();

    System.out.println("In attesa che i threads abbiano terminato.");
    // Wait for all threads to complete
    try {
      t1.join();
      t2.join();
    } catch (InterruptedException e) {
      /* Unhandled exception */
    }
    System.out.println("Tutti i threads hanno terminato.");
  }
}
```

Soluzione esercizio 3

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

class Worker implements Runnable {
  private int[] values;
  private int start;
  private int length;

  public Worker(int[] values, int start, int length) {
    this.values = values;
    this.start = start;
    this.length = length;
  }

  @Override
  public void run() {
    int sum = 0;
    for (int i = 0; i < length; i++)
      sum += values[start + i];
    System.out.println("Somma [" + start + " ; " + (start + length - 1) + "] = " + sum);
  }
}

public class S1Esercizio3 {
  final static int ARRAYSIZE = 10000;
  final static int NUM_THREADS = 10;

  public static void main(String[] args) {
    int length = ARRAYSIZE / NUM_THREADS;
    List<Worker> allWorkers = new ArrayList<Worker>();
    List<Thread> allThreads = new ArrayList<Thread>();

    Random r = new Random();
    int[] array = new int[ARRAYSIZE];
    for (int i = 0; i < array.length; i++)
      array[i] = r.nextInt(100) + 1;

    for (int i = 0; i < 10; i++) {
      /* Crea worker e thread */
      Worker worker = new Worker(array, (i * length), length);
      allWorkers.add(worker);
      allThreads.add(new Thread(worker));
    }

    /* Fa partire i thread */
    for (Thread t : allThreads)
      t.start();

    /* Attende che tutti i thread terminano */
    for (Thread t : allThreads) {
      try {
        t.join();
      } catch (InterruptedException e) {
        /* Unhandled exception */
      }
    }
  }
}
```