Esercizio 1

Utilizzando la tecnica one ***writer many readers***, troviamo un solo thread (nel nostro caso il *main-thread*) in grado di scrivere (e dunque cambiare) lo stato dell'oggetto condiviso mentre tutti gli altri sono liberi di leggerlo.

Per implementare la tecnica, le variabili *isRunning* e *sharedValue* (nella classe che contiene il *main()*) devono essere modificate da ***public*** a ***private*** e creare dunque i *getter* necessari per accedere ai valori di queste variabili, in modo che possano essere modificate solamente dalla classe del *main-thread*. Nel metodo *run()* degli lettori bisogna invece (oltre ad eliminare i lock come richiesto) memorizzare il valore della variabile condivisa *sharedValue* in una variabile condivisa: per i successivi utilizzi della variabile verrà utilizzata la copia locale e non più la vera variabile condivisa, in modo da ridurre le possibilità di utilizzare valori sporchi o scorretti.

```
public void run()
{
	while (Esercizio1.getIsRunning()) {
		try {
			int newValue=Esercizio1.getSharedValue();
			// Update local value if needed
			if (localValue != newValue)
				localValue = newValue;
			else
				System.out.println("Reader" + id + ": (" + localValue
					+ " == " + newValue + ")");
		} finally {

		}
	}
}
```

Esercizio 2

Per implementare la classe in modo da creare oggetti *immutable*, è necessario che l'oggetto, una volta creato, mantenga il suo stato invariato, quindi non modificabile da nessun altro metodo/oggetto per tutta la durata della sua esistenza.

Per prima cosa è necessario rendere le variabili d'istanza di tipo *final* e con modificatore di visibilità *private*, in modo che queste non siano accessibili dall'esterno e vengano inizializzate solo una volta (durante la loro costruzione) dal costruttore. In seguito, per poter accedere al valore di questa variabile, bisogna creare i metodi *getter* necessari.

Il costruttore deve occuparsi di assegnare il corretto valore alle variabili di cui sopra, ricevendoli in ingresso attraverso i corretti parametri. Per rendere inoltre la classe non ereditabile, è possibile seguire due strade diverse: la prima, aggiungendo all'intestazione della classe il modificatore *final* oppure, come in questo caso, dichiarare il costruttore come *private* ed aggiungere un metodo statico (quindi con modificatore *static*) che riceve in ingresso i parametri e richiama il costruttore della classe, restituendo un nuovo oggetto del tipo della classe.

Come ultima cosa bisogna assicurarsi che eventuali oggetti di tipo referenza non vengano in qualche modo esposti dalla classe (esempio: metodo che ritorna una lista, potrebbe esporre il suo indirizzo e quindi renderla non più *immutable*).

Il metodo *run()* dei diversi thread deve essere leggermente modificato per spostare l'assegnamento dei valori interni dell'oggetto *immutable* durante la fase di creazione dell'oggetto.

Variabili d'istanza

```
private final double lat;
private final double lon;
```

Metodo statico per costruire un nuovo oggetto

```
public static Coordinate CreaCoordinate(final double lat, final double lon)
{
        return new Coordinate(lat, lon);
}
```

Codice che sostituisce la creazione e il successivo assegnamento dei valori all'oggetto immutable

```
Coordinate temp = Coordinate.CreaCoordinate(ThreadLocalRandom.current().nextDouble(-90.0, +90.0), ThreadLocalRandom.current().nextDouble(-180.0, +180.0));

Esercizio2.lock.lock();
try {
        Esercizio2.curLocation = temp;
} finally {
        Esercizio2.lock.unlock();
}
```

Inoltre, per risolvere eventuali problemi di visibilità, potrebbe essere una buona soluzione rendere la variabile condivisa *completed* di tipo *volatile* oppure cambiare totalmente il suo tipo in una variabile atomica di tipo *AtomicBoolean*.