Domanda **1**

Risposta corretta

Punteggio max.: 1,00

Contrassegna domanda

Associare ciascuna struttura di gestione della concorrenza al suo ambito di applicazione:

`Semaphore`	Gestione di un insieme omogeneo di risorse. ✔
`Object::wait()`	Gestione esplicita dell'accesso ad un singolo oggetto. ✔
`syncronized`	Gestione della concorrenza tramite la struttura sintattica del codice. ✔
`Condition`	Gestione dell'accesso alla stessa sezione critica in condizioni di blocco e/o sblocco differenti. ✔
`Lock`	Gestione esplicita, senza legami sintattici, del blocco e dello sblocco della sezione critica. ✔

Domanda **2**

Risposta corretta

Punteggio max.: 1,00

Contrassegna domanda

Implementare un programma di rete usando l'astrazione dei `Channels` della libreria standard di Java permette di non occuparsi di molti dettagli riguardanti l'interazione con il mezzo di comunicazione, ma:

- E' sufficiente sostituire il codice che gestisce la ricezione di un pacchetto di dati.
- E' sufficiente sostituire il codice che gestisce una nuova connessione entrante.
- E' necessario riscrivere le parti che interagiscono con il mezzo di comunicazione per gestire in modo diverso la concorrenza.
- (●) E' necessario ristrutturare il nostro codice riorganizzando in metodi che vengono richiamati all'avvenire di specifici eventi di I/O. ✔

Domanda **3**

Risposta corretta

Punteggio max.: 1,00

Contrassegna domanda

L'astrazione `Channel` della libreria standard di Java permette di non occuparsi di molti dettagli riguardanti l'interazione con il mezzo di comunicazione. I vari metodi che l'astrazione richiede di implementare sono accomunati dall'uso di un parametro `attachment`. Il suo scopo è:

- (●) Trasportare in sicurezza il contesto della conversazione, fra metodi che saranno richiamati da `Thread` differenti. ✔
- Distribuire fra i vari metodi i dati globali del programma.
- Fornire il dato letto dal canale di comunicazione.
- Rendere accessibile in ogni momento il canale sottostante la comunicazione.

Domanda **4**

Risposta corretta

Punteggio max.: 1,00

Contrassegna domanda

In questo codice di esempio:

```
try(DatagramSocket socket= new DatagramSocket(8080)) {
  byte[] buf = new byte[16];
  DatagramPacket packet = new DatagramPacket(buf, 16);
  // A
  String input = new String(packet.getData(), 0, packet.getLength());
  System.out.println(input);
  // B
}
```

in che punto va inserita la chiusura della risorsa DatagramSocket:

- Nel punto B.
- Non è necessaria. ✔
- In un blocco finally da aggiungere in coda al blocco try.
- Nel punto A.

Domanda **5**

Risposta corretta

Punteggio max.: 1,00

Contrassegna domanda

Lo scopo delle Reactive Extensions è:

- Fornire una API per definire elaborazioni di sequenze di oggetti.
- Fornire una semantica per definire elaborazioni asincrone di sequenze di oggetti. ✔
- Fornire un modello di esecuzione di elaborazioni parallele di insiemi di oggetti.
- Fornire un insieme di componenti per l'elaborazione distribuita di stream di valori.

Domanda **6**

Risposta corretta

Punteggio max.: 1,00

Contrassegna domanda

Quando si dice che il compilatore Java ha delle capacità di Type Inference si intende che:

- E' in grado di indicare se il grafo dell'ereditarietà genera un *diamond problem*.
- E' in grado di calcolare la corretta indentazione del codice e correggerla.
- E' in grado di trasformare un tipo in un'altro senza indicazioni esterne.
- E' in grado di dedurre il tipo di alcune espressioni senza che sia necessario indicarlo esplicitamente. ✔

Domanda **7**

Risposta corretta

Punteggio max.: 1,00

Contrassegna domanda

Perché nel linguaggio Java si è deciso di introdurre i metodi di default nelle Interfacce?

- Per rendere più difficile modificare l'implementazione delle interfacce in modi non previsti.
- Per evitare una possibilità di realizzare un *diamond problem*.
- Per inseguire una feature richiesta dal mercato.
- Per poter estendere delle interfacce consolidate senza richiedere l'aggiornamento del codice esistente. ✔

Domanda **8**

Risposta corretta

Punteggio max.: 1,00

Contrassegna domanda

Una operatore *short-circuiting* all'interno di una catena di elaborazione di uno Stream può:

- Ottenere uno Stream infinito da una funzione di trasformazione.
- Rendere seriale l'elaborazione di uno Stream parallelo.
- Produrre il risultato prima che lo Stream sia stato interamente consumato. ✔
- Cambiare l'ordine degli elementi dello Stream.

Domanda **9**

Risposta errata

Punteggio max.: 1,00

Contrassegna domanda

Usando i Reactive Stream, la gestione più granulare della composizione della pipeline di elaborazione dello stream permette di:

- Distribuire i singoli componenti dell'elaborazione su più nodi, indicando su quali nodi aumentare il parallelismo e su quali accumulare i risultati da elaborare serialmente. ✖
- Scegliere algoritmi di suddivisione del lavoro più efficienti di quelli della libreria standard, perché più facili da aggiornare.
- Ridurre la latenza dell'elaborazione dei vari componenti decidendo quante risorse dedicare a ciascuno di essi.
- Isolare la parte di pipeline che si desidera sia resa parallela; gli Stream della libreria standard sono o completamente paralleli, o completamente seriali.

Domanda **10**

Risposta corretta

Punteggio max.: 1,00

Contrassegna domanda

Se in un sistema distribuito i nodi non trovano un consenso sullo stato del sistema, può accadere che:

- Le risposte del sistema siano molteplici e conflittuali perché raccolgono i dati da più nodi.
- Le risposte del sistema siano inefficienti perché le differenti versioni dello stato si accavallano in una `race condition`.
- Le risposte del sistema non siano disponibili in quanto gli stati differenti si annullano.
- Le risposte del sistema siano incoerenti e dipendano da quale nodo viene contattato. ✔

Domanda **11**

Risposta corretta

Punteggio max.: 1,00

Contrassegna domanda

Quale delle seguenti affermazioni riguardo ai rapporti fra Processi, Thread e Fiber è vera:

- Le risorse dei Processi sono controllate dal Sistema Operativo, mentre all'interno dei Processi i Thread devono direttamente controllare il loro accesso. Le Fiber rendono esplicita la concorrenza con lo scopo di essere ancora più leggere dei Thread. ✔
- I Processi evitano il deadlock attraverso l'ordinamento delle priorità. I Thread invece non sono ordinati e devono essere manualmente controllati per evitare conflitti nella gestione delle risorse. Le Fiber sono una evoluzione più efficiente dei Thread.
- I Processi sono raggruppamenti logici di risorse che nei Thread vengono associati a risorse fisiche. Le Fiber sono un miglioramento dei Thread.
- Una volta allocata una risorsa, non può essere sottratta ad un Processo. Ad un Thread invece può essere sottrratta, mettendolo in stato "waiting". Le Fiber rendono la gestione della concorrenza più esplicita.

Domanda **12**

Risposta corretta

Punteggio max.: 1,00

Contrassegna domanda

Nell'astrazione delle Reactive Extensions, un `Subject` può:

- Osservare altri `Subject` e `Observable` alterando la struttura dello stream fra di loro.
- Osservare uno Stream in modalità parallela.
- Osservare diversi `Observable`, e comportarsi da `Observable` esso stesso, modificando la struttura dell'elaborazione dello stream. ✔
- Osservare uno Stream esaminando solo alcuni elementi.

Domanda **13**

Risposta corretta

Punteggio max.: 1,00

Contrassegna domanda

Un oggetto `Future` rappresenta:

- (•) Un calcolo che potrebbe produrre un risultato dopo un certo tempo. ✔
- ( ) Il risultato di un calcolo parallelo terminato correttamente.
- ( ) Un calcolo concorrente terminato in modo errato.
- ( ) Una generica esecuzione concorrente.

Domanda **14**

Risposta corretta

Punteggio max.: 1,00

Contrassegna domanda

Quale di queste caratteristiche è propria della sintassi `switch-case` come espressione:

- ( ) E' possibile il *fall-through* da un caso all'altro.
- ( ) Ogni caso deve produrre un risultato diverso.
- (•) L'elenco delle opzioni deve essere esaustivo. ✔
- ( ) I risultati devono essere tutti valori della stessa interfaccia.

Domanda **15**

Risposta errata

Punteggio max.: 1,00

Contrassegna domanda

Date le seguenti classi:

```
class Foo
  Foo(int a) {
    // Costruttore Foo
  }
}

class Bar extends Foo {
  static {
    // Inizializzatore statico
  }
  {
    // Inizializzatore
  }
  Bar(int a, String b) {
    super(a);
    // Costruttore Bar
  }
}
```

Ordinare le strutture indicate secondo la sequenza con cui verranno eseguite: { =Inizializzatore statico->1 =Costruttore Foo->2 =Inizializzatore->3 =Costruttore Bar->4 }

Risposta: 1,2,4,3

Inizializzatore statico 1, Costruttore Foo 2, Costruttore Bar 4, Inizializzatore 3