Soluzioni master parte teorica

Domanda 1:

- 1 punto: tipi valore sono int double long float … (almeno 4)
- 1 punto: i tipi valore vengono passati per valore
- 1 punto: tipi referenza sono tutti quelli definiti da class o interface
- 1 punto: i tipi referenza sono tutti sottotipo di Object
- 1 punto: i tipi referenza vengono passati per referenza/indirizzo e creano quindi problematiche di aliasing

Domanda 2:

- 1 punto: quando lanciata, un'eccezione termina il flusso sequenziale di esecuzione ed entra in una sorta di stato di errore
- 1 punto: classe che estende Exception/Throwable
- 1 punto: eccezioni lanciate con throw
- 1 punto: try{…} catch(E e) {…} cattura eccezioni di tipo E o sottotipo lanciate dal blocco nel try
- 1 punto: finally viene sempre eseguito, se eccezione poi si riprende in stato "eccezionale"

Domanda 3:

- 1 punto: le reflection permette di vedere la struttura del programma (classi, metodi, campi, …)
- 1 punto: class Class, Method, Field
- 1 punto: getField/Method/Constructor
- 1 punto: setField per assegnare un campo, invoke method
- 1 punto: vedere come il codice e' annotato

Soluzioni master parte pratica

Esercizio 1:

- 1 punto: campo codicefiscale o getter
- 1 punto: tutti i campi di Persona sono final, o sono private e assegnati solo nel costruttore
- 2 punto: classe PersonaVisitata extends Persona oppure contiene una Persona (1 punto) e getter della Persona contenuta (1 punto)
- 1 punto: la classe PersonaVisitata contiene i due campi priorita' e reparto
- 1 punto: i campi sono accessibili in lettura in qualche modo (pubblico o getter)
- 1 punto: i campi sono accessibili in scrittura in qualche modo (pubblico o setter)

Esercizio 2:

- 1 punto: interfaccia Reparto
- 1 punto: due metodi di visita se PersonaVisitata non estende Persona, un metodo altrimenti, con le firme dei metodi corrette
- 1 punto: il metodo ritorna la persona non modificata (se rientra nel caso) correttamente
- 1 punto: la persona e' modificata correttamente se la priorita' e' maggiore di 100

Esercizio 3:

- 1 punto: campo Collection<Reparto> (o simili se inizializzati in maniera opportuna)
- 1 punto: ciclo foreach (o simile) che scorre tutti i reparti
- 1 punto: controllo del nome del reparto corretto (con .equals e non ==)
- 1 punto: corretta invocazione e ritorno del risultato del metodo di Reparto