Esercizio 1 — Gestione di arnie (ereditarietà, incapsulamento, overload, is/as, ref/out)

Obiettivo: implementare una gerarchia di classi per rappresentare arnie in un apiario, dimostrando ereditarietà, incapsulamento (private/protected/public), overload di costruttori e metodi, uso di collezioni di oggetti (`List<T>`), operatori `is`/`as`, e tre diversi modi di passare parametri in C# (valore, `ref`, `out`).

Specifiche (classi richieste)

1. **Classe base `Arnia` (public)**
   - Campi/proprietà:
     - `private int id;`
     - `protected string posizione;` // es.: “Apiario Nord”
     - `public double quantitaMiele { get; private set; }`
   - Costruttori (overload):
     - `Arnia(int id)` — imposta `posizione` a `"Sconosciuta"`.
     - `Arnia(int id, string posizione)` — imposta `posizione`.
   - Metodi:
     - `public virtual void AggiungiMiele(double kg)` — aggiunge `kg` a `quantitaMiele`.
     - `public string GetPosizione()` — ritorna `posizione`.
     - `public int GetId()` — ritorna `id`.
   - Regole di incapsulamento: `id` è private (non accessibile dalle sottoclassi), `posizione` è protected (accessibile dalle sottoclassi).
2. **Sottoclasse `ArniaIntelligente` (deriva da `Arnia`)**
   - Campi/proprietà:
     - `private double batteria;` (percentuale)
   - Costruttori (overload):
     - `ArniaIntelligente(int id)` — usa il costruttore base.

        - `ArniaIntelligente(int id, string posizione, double batteria)` — imposta tutti i campi.
    - Metodi:
        - `public override void AggiungiMiele(double kg)` — oltre ad aggiungere il miele chiama il metodo base e registra (console) l'operazione (es.: log).
        - `public void AggiungiMiele(int kg, bool urgente)` — **overload**: versione con firma diversa (int, bool) che converte e chiama la versione `double`.
        - `protected void RiportaStatoBatteria()` — stampa la batteria (dimostra protected/ accesso).
        - `public void DecrementaBatteria(ref double consumo)` — dimostra `ref`: riduce la batteria usando il valore passato per riferimento e aggiorna `consumo`.
        - `public void ResetBatteria(out double nuovaBatteria)` — dimostra `out`: imposta batteria a 100 e comunica il valore tramite `out`.

Requisiti funzionali (comportamento richiesto)

- Creare una `List<Arnia>` e inserire in essa sia oggetti `Arnia` che `ArniaIntelligente`.
- Iterare la lista e, per ogni elemento:
    - Usare `is` per verificare se è `ArniaIntelligente`.
    - Se lo è, usare `as` per cast sicuro e chiamare `DecrementaBatteria` passando un `double` come `ref` e poi `ResetBatteria` con `out`.
    - Chiamare `AggiungiMiele` con valori diversi (doppio / intero) per mostrare l'overload.
- Implementare un metodo `static void SpostaArnia(Arnia a)` che prova a riassegnare il parametro `a` a un nuovo oggetto — documentare (via commento e test) come **la riassegnazione non modifica** la variabile originale del chiamante (passaggio per valore del riferimento).
- Implementare un metodo `static void SostituisciArniaRef(ref Arnia a)` che sostituisce l'oggetto passato (dimostrare `ref` a livello di riferimento stesso).
- Implementare test in `Main()` che mostrino i risultati delle chiamate sopra (stampe su console).

Esempio di esecuzione attesa (console)

```
Creata Arnia id=1 posizione=SitoA mieli=0
Creata ArniaIntelligente id=2 posizione=SitoB batteria=85
Aggiunto 2.5 kg a Arnia id=1 -> totale 2.5
Aggiunto 1 kg a ArniaIntelligente id=2 -> totale 1.0 (override + log)
Aggiunto 3 kg (int, urgente) a ArniaIntelligente id=2 -> totale 4.0
Arnia id=2 è ArniaIntelligente -> decremento batteria con ref: consumo iniziale 5 -> consumo dopo chiamata 3
Reset batteria effettuato: nuovaBatteria=100
Tentativo di sostituire senza ref: la variabile originale rimane id=1
Sostituzione con ref: la variabile originale ora è id=99
```

(I messaggi devono provenire dalle chiamate implementate; i valori sono indicativi.)

Criteri di valutazione e punti obbligatori

- Correctness: ereditarietà e override devono funzionare (10 pt).
- Incapsulamento: uso corretto di `private`, `protected`, `public` e accessibilità (10 pt).
- Overload metodi e costruttori (10 pt).
- Uso `List<Arnia>`, `is` e `as` per distinguere tipi e chiamare metodi specifici (10 pt).
- Dimostrazione dei 3 modi di passaggio (passaggio riferimento oggetti per valore, `ref`, `out`) e spiegazione (10 pt).
- Bonus: aggiungere eccezioni gestite (es. non permettere miele negativo), test unitari o commenti esplicativi (fino a 5 pt).