Cognome: **Nome:** **Matricola:** **Anno Progetto Didattico:**

Note:

- Tutti i diagrammi disegnati devono utilizzare la sintassi del linguaggio UML 2.x ed essere opportunamente commentati.
- Il foglio protocollo va utilizzato solo per la brutta copia. Per le risposte, redatte a penna e non a matita, vanno utilizzati gli spazi liberi, fronte e retro, della corrispondente pagina.
- Riportare il proprio nome, cognome, matricola, e anno di progetto didattico su tutti i fogli consegnati.

Punti 6/30

Il sito *online* di una banca è parte dei suoi sistemi di "multicanalità". Attraverso di esso, è possibile reperire informazioni generiche ed effettuare operazioni dispositive. Per essere riconosciuto dal sistema, un utente deve inserire il proprio codice titolare, un PIN, e un codice alfanumerico casuale, solitamente generato *ad hoc* da un *token* fisico. All'interno della propria pagina di ingresso personalizzata, l'utente può visualizzare la lista di movimenti effettuati sul proprio conto corrente negli ultimi 30 giorni. Per ogni movimento, tale lista riporta la categoria dell'operazione (pagamento stipendio/pensione, operazione acquisto alimentari, pagamento F24, ...), la sua data e il suo importo. Se il movimento è un trasferimento (con prelievo), ne viene visualizzato il beneficiario, altrimenti il pagatore. Per facilitare l'utenza nel consultare i propri movimenti, il sito permette diversi tipi di ricerca: per beneficiario, per tipologia di movimentazione, per importo, per periodo. L'utente può inoltre associare una propria carta di credito alla propria pagina personale, inserendone il numero, la data di scadenza e il codice CVV2. Un errore informa l'utente se le informazioni inserite non sono corrette. Il sito della banca offre ovviamente anche funzioni per l'utenza generica, come ricercare filiali della banca per CAP o città. È inoltre disponibile una sezione informativa, che descrive i principali prodotti della banca, quali mutui, prestiti e polizze assicurative.

Si utilizzino i diagrammi dei casi d'uso per modellare gli scenari sopra descritti. Non ne è richiesta la descrizione testuale.

Cognome: **Nome:** **Matricola:** **Anno Progetto Didattico:**

Punti 7/30

La programmazione moderna è sempre più orientata all'utilizzo del "modello a eventi", detto *reactive programming*. In esso, anziché controllare attivamente se l'evento atteso sia disponibile, si designa una risorsa del programma a venirne direttamente e immediatamente notificata. Le strutture che gestiscono gli eventi sono dette *stream*. Uno *stream* può essere di uno di tre tipi: uno *stream* di tipo `Source` genera eventi; uno *stream* di tipo `Sink` consuma eventi; uno *stream* di tipo `Flow` consuma un evento e ne genera un altro, in cascata. Invece che essere staticamente associate a uno di tali tipi dedicati, le strutture *stream* sono generiche sui tipi. Pertanto, un oggetto di tipo `Source` può essere osservato da oggetti di tipo `Sink` o `Flow`, mentre un oggetto di tipo `Flow` può essere osservato da oggetti di tipo `Sink`. `Source` e `Flow` possono locali al programma oppure remoti. Per mantere distinta la logica di controllo degli *stream* e l'implementazione della loro logica di *business*, ogni tipo di *stream* ha associato un tipo `*Action`, quindi `SouceAction`, `FlowAction`, e `SinkAction`. Ognuno di tali tipi espone un unico metodo per implementare il comportamento desiderato dello *stream*.

Si modelli il sistema descritto mediante un diagramma delle classi che utilizzi opportuni *design pattern*. Utilizzando un diagramma di sequenza, si descriva poi una catena di *stream* che, leggendo, in porzioni successive, un *file* di testo da un *file system* remoto, ne stampi il contenuto su una *console*.

Cognome: Nome: Matricola: Anno Progetto Didattico:

Punti 3/30

Si considerino le seguenti classi, che permettono di copiare semplici caratteri immessi da tastiera, direttamente verso una stampante.

```
public class Copier {
    private KeyboardReader reader;
    private PrinterWriter writer;
    public Copier(KeyboardReader reader, PrinterWriter writer) {
         this.reader = reader;
         this.writer = writer;
    }
    public void copy(OutputDevice dev) {
        char c;
        while ((c = this.reader.read()) != EOF)
            if (dev == PRINTER)
                writer.write(c);
    }
}
public class KeyboardReader {
    public char read() {
        // Reads a character from the keyboard
    }
}
public class PrinterWriter {
    public void write(char c) {
        // Writes the character directly into the printer spool
    }
}
```

Si fornisca il diagramma delle classi di una soluzione che permetta di estendere tali classi in modo che la classe `Copier` possa essere riutilizzata per trasferire caratteri provenienti da un *Socket,* direttamente su un *file* residente all'interno di un *file system* HDFS.

La soluzione dovrà tenere in considerazione il *Dependency Inversion Principle*.

Cognome: Nome: Matricola: Anno Progetto Didattico:

Punti 5/30

Fornire una definizione di "requisito", applicabile al dominio dell'ingegneria del *software,* e descrivere, succintamente ma con precisione, il ciclo di vita dei requisiti all'interno di un progetto del tipo di quello didattico, rappresentandolo come una apposita macchina a stati, specificando anche le attività poste sugli archi di transizione.

Punti 4/30

L'esercizio 3 di questo compito chiede di applicare il *Dependency Injection Principle*. Fornirne una definizione che ne illustri le differenze rispetto allo stile architetturale "a livelli" (p.es. quello applicato nella pila dei protocolli OSI).

Punti 5/30

Fornire una definizione del concetto di *framework*, specificamente collocato in relazione alla progettazione *software*, insieme a una descrizione sintetica dei criteri applicati per selezione ed eventuale uso di qualcuno di essi, come sperimentato nel proprio progetto didattico.