# Assigment 2

## Descrizione del progetto

Implementare un convertitore per i primi 1000 numeri interi da arabi a romani. Implementare un metodo per stampare i numeri romani in asciart:

```
 _____  __      __ __   __  _         _____   _____    __  __
|_   _| \ \    / / \ \ / / | |       / ____| |  __ \  |  \/  |
  | |    \ \  / /   \ V /  | |      | |      | |  | | | \  / |
  | |     \ \/ /     > <   | |      | |      | |  | | | |\/| |
 _| |_     \  /     / . \  | |____  | |____  | |__| | | |  | |
|_____|     \/     /_/ \_\ |______|  \_____| |_____/  |_|  |_|
```

Il programma deve soddisfare i requisiti descritti a questo indirizzo.

Si consiglia di implementare il progetto seguendo il seguente ordine:

- Convertire e stampare i primi 3 numeri interi
- Convertire e stampare i primi 6 numeri interi
- Convertire e stampare i primi 10 numeri interi
- Convertire e stampare i primi 20 numeri interi
- Convertire e stampare i primi 50 numeri interi
- Converitre e stampare i primi 100 numeri interi
- Convertire e stampare i primi 500 numeri interi
- Convertire e stampare i primi 1000 numeri interi

## Interfaccia e oggetti

Sviluppare il progetto implementando le seguenti classi possibilmente con l'approccio Test Driven Development

```
package it.unipd.mtss;

public class IntegerToRoman {

    public static String convert(int number){
        // TODO
        return null;
    }
}
```

```
package it.unipd.mtss;

public class RomanPrinter {
    public static String print(int num){
        return printAsciiArt(IntegerToRoman.convert(num));
    }

    private static String printAsciiArt(String romanNumber){
        //TODO
        return null;
    }
}
```

# Svolgimento

Sviluppare il progetto seguendo la pratica della Continuous Integration, pertanto si richiede di svolgere l'assignment nel seguente ordine:

- Predisposizione del repository github
- Definizione delle attività nel Issue Traking System
    - Suddividere le attività tra i due sviluppatori
- Creazione del progetto utilizzando l'archetipo maven quick start
- Configurazione del processo di build
- attivazione e configurazione dell'analisi statica
- Predisposizione della Build Automation con GitHub Actions
- Sviluppo del codice sorgente e dei test

Di seguito vengono specificate in dettaglio le configurazioni dei vari strumenti.

## Issue Traking System

Gestire il progetto nel issue tracking system di github:

- registrare la nuova versione (vedi laboratorio ITS: Iterazioni o Milestone)
- creare la project board (vedi laboratorio 1: Bacheche (Project Board))
- creare le attività per la realizzazione del assigment 2
- suddividere le attività tra i due sviluppatori

## Version Control System

Sviluppare il progetto utilizzando un repository git pubblico in Github.

Le attività possono essere gestite utilizzando il work flow *Feature Branch*.

Le attività devono essere sviluppate (compresi i test di unità) una per volta senza tenere in considerazione le attività successive.

# Project Automation

- Creare il progetto con Maven utilizzando l'archetipo: `maven-archetype-quickstart`
- Utilizzare i seguenti parametri:
  - **groupId**: it.unipd.mtss
  - **artifactId**: roman-number

Configurare il Build Lifecycle in modo da:

- eseguire la compilazione del progetto (fase compile)
- eseguire i test di unità (fase test)
- eseguire l'analisi statica del codice con checkstyle nella fase **package** (Vedi laboratorio Maven: Plugin - Universal reuse of business logic).

# Test di Unità

- Sviluppare i test di unità per arrivare ad una copertura del code >= 85% dei sorgenti di produzione
- I test di unità devono essere sviluppati con il framework JUnit 4 o 5 e seguire le convenzioni Maven (vedi qui e qui)
- Le firme dei metodi di test devono essere parlanti. Vedi il formalismo riportato nel laboratorio JUnit. I test devono essere sviluppati seguendo il pattern Arrange/Act/Assert (AAA)
- Creare dei test di unità per soddisfare le caratteriste **A-TRIP** e **Right Bicep** viste a lezione. È richiesta la copertura del codice >= 85%.
- [Opzionale] sviluppare il progetto utilizzando l'approccio TDD
- [Opzionale] Utilizzare il framework Mokito per creare i test per la classe RomanPrinter

# Analisi statica del codice

Configurare il plugin maven checkstyle in modo da effettuare le seguenti verifiche al codice di produzione (creare il file `checkstyle.xml` e configurare i seguenti moduli):

- BooleanExpressionComplexity
- CyclomaticComplexity
- FileLength
- LineLength
- MethodLength
- EmptyCatchBlock

- FileTabCharacter
- AvoidStarImport
- IllegalImport
- NeedBraces
- Header con il seguente valore (sostituire le variabili con i vostri dati)

```
////////////////////////////////////////////////////////////////
// [NOME1] [COGNOME1] [MATRICOLA1]
// [NOME2] [COGNOME2] [MATRICOLA2]
////////////////////////////////////////////////////////////////
```

Configurare il plugin in modo da far fallire la build se non vengono rispettati i controlli configurati.

## Build Automation

- Attivare la Build Automation con GitHub Action.
- Pubblicare il risultato del processo di build nella pagina README.md del progetto
- Attivare il caloclo del code coverage

> **TIP** vedi Jacoco e Coveralls

- [Opzionale] Pubblicare il risultato dell'analisi statica e del code coverage nella pagina README.md del progetto

# Modalità di consegna

- L'assignment deve essere **completato e consegnato entro il 18/05/2022 alle 18:00**. Ogni consegna e modifica successiva non sarà valutata.
- Il codice prodotto deve essere rilasciato nel ramo **master** o **main**
- La consegna deve essere fatta tramite l'apposito form che verrà pubblicato nel sito del corso