# Assignment A1

## Assignment A1

| | |
|---|---|
| Released by: | Monday, 28 July 2025, 09:00 |
| Base assignment deadline: | Friday, 08 August 2025, 15:00 |
| Code Walk registration/preferences/ext. tokens deadline: | Friday, 08 August 2025, 18:00 |
| Code Walk registration link: | (link) |
| Code Walks: | Wednesday, 13 August 2025<br>Thursday, 14 August 2025<br>Friday, 15 August 2025 |
| GitLab template repository: | (link) |
| Compliance tests JAR file: | JAR file |
| Minimum Library Version: | 2025S1-7 |
| Last Updated: | Sunday, 03 August 2025, 20:53 |

This assignment is restricted to using the Functional Java libraries. Please note the required minimal library version above! In everything you do, you must follow the Design Recipe. This includes adding **documentation** to all functions created, *except* for the functions used for testing (that is, testing interfaces) and test functions. To see roughly how this assignment will be marked, check out the Skeleton Rubric.

> You really really need to stick to the restrictions of Functional Java. Don't just write Java code simply because you already know Java. The penalties in the rubric for doing so are quite harsh.

## Formalities

This assignment is submitted via GitLab.

## Access to Your Repository

To start your assignment, go to the GitLab repository of Assignment 1 and follow these steps:

1. Click the Fork button at the top right of the page.
2. Do not change the project name, that is, it must still be `comp1110-2025s2-a1`
3. If asked for a namespace, choose your university ID.
4. Do not change the project slug.
5. Make sure the visibility of the project (bottom of the page) is set to **private**.
6. Click the `Fork project` button at the bottom of the page.

To double check that we can access your assignment, make sure that:

- your assignment 1 project is reachable through the address `https://gitlab.cecs.anu.edu.au/XXXXXXXX/comp1110-2025s2-a1` where **XXXXXXXX** is your uid in the format `u1234567` .
- your repository includes our marker bot ( `comp1110-2025-s2-marker` ) as a `Maintainer` member (click on `Manage` on the right side of your project page, then `Members` and you should see the marker bot in the list on the right).

> You can achieve both these points by creating your project through forking our template repository and not changing the name, or by just using the correct name when creating the project and then manually adding the bot.

## Notebooks and Push Requirement

You need to keep a notebook file as explained on the Git and Notebooks page. You need to commit *and* push the current state of your files:

- at the end of every session where you work on the assignment
- at least once per hour (our checks implement that no two pushes during sessions recorded in the notebook are more than 70 minutes apart).
- at least once per major part of the assignment (i.e. for this assignment, at least four times). You are absolutely free to commit and/or push more often than that.

You need to write informative commit messages, and informative notes for each session in your notebook, which describe what it is what you did for this particular commit or session, respectively.

## Statement of Originality

Your submission (and therefore your git repository) needs to include a signed statement of originality. This must be a text file named `soo.txt`. That file must contain the following text, with the placeholders `[your name here]` and `[your UID here]` replaced with your name and UID, respectively:

```
I declare that this work upholds the principles of academic
integrity, as defined in the University Academic Misconduct
Rule; is entirely my own work; is produced for the purposes
of this assessment task and has not been submitted for
assessment in any other context, except where authorised in
writing by the course convener; gives appropriate
acknowledgement of the ideas, scholarship and intellectual
property of others insofar as these have been used; in no part
involves copying, cheating, collusion, fabrication, plagiarism
or recycling.

I declare that I have not used any form of generative AI in
preparing this work.

I declare that I have the rights to use and submit any assets
included in this work.

[your name here]
[your UID here]
```

## Files Summary

Your repository *must* be accessible to us and *must* contain the following files in its root directory:

- `notebook.yml` -your Notebook
- `soo.txt` -your **signed** Statement of Originality

**The deadline for registering to participate in code walks is Friday, 08 August 2025, 18:00. You need to do this in order for your assignment submission to get marked!**

## Compliance Tests

To assist you with all the formalities above and the required records and functions defined in the following sections, use the A1Compliance.jar checker. Save the compliance checker to the same folder as your assignment and execute the following in the terminal to check your assignment: `java -jar --enable-preview A1Compliance.jar .`

The current version of the A1 compliance checker is 1.01. Always ensure that you have the latest version of it by verifying that the version displayed in the first output line matches the previously mentioned one.

Keep in mind that **passing the compliance tests does not imply that you will get a good grade because the compliance checker does not evaluate the logic of your functions.**

# Tasks

In Parts 1 to 3, you will model interstellar scavengers that find and trade artifacts recovered from the depths of space. All parts are to be completed in a single file called `Artifacts.java`.

There are 3 categories of artifacts and various types of scavengers, including rational agents and risk-taking agents, which we will model. Each scavenger differs in how they evaluate these artifacts and uses a decision-making process to determine whether a newly found artifact is more valuable than the one they currently possess. When analysing two artifacts, scavengers will classify the new artifact in relation to the owned one and get a result that can be either `VALUABLE`, `MUNDANE`, `HAZARDOUS`, `INCOMPATIBLE`, or `UNKNOWN`.

## Part 1: Artifact Analysis

*Part 1 is a regular part of the assignment and will be included in marking it. However, it is designed as an on-ramp. Tutors may help you with the code here if you ask for help - thoug you should still try to do as much as possible yourself.*

Each artifact falls into one of three categories:

- A **Star Chart**: A complex navigational tool. It is defined by its destination (`String`), a calculated risk factor (`int`), and its origin point, defined by a `sector` and `system` (both `int`s).
- An **Energy Crystal**: A simple power source, defined only by its power level (`int`).
- A **Inert Rock**: An inert and mysterious object, defined only by its `color` (`String`).

### Rational Scavenger

You will design a function representing how a rational scavengers decides which artifact they prefer between an artifact they have and a new artifact. The analysis rules are as follows:

- When comparing two **Star Charts**, if the new star chart has a *lower* risk factor than the star chart they own, then the new star chart is considered `VALUABLE` otherwise it is considered `MUNDANE`.

- When comparing two **Energy Crystals**, if the new crystal has a *higher* power level than the crystal they own, the new crystal is considered `VALUABLE`; otherwise, it is `MUNDANE`.
- When comparing two **Inert Rocks**, if they are the same color, the new rock is considered `MUNDANE`. If their colors are different, they are `INCOMPATIBLE`.
- When comparing a **Star Chart** with an **Energy Crystal**, the result of the analysis depends on which one is the new artifact. If the new artifact is a Star Chart, the result is `HAZARDOUS` due to the risks of travel. If the new artifact is an Energy Crystal, the result is `MUNDANE`.
- Any comparison involving an **Inert Rock** and a different type of artifact (a Star Chart or an Energy Crystal) yields an `UNKNOWN` result, as their value cannot be determined relative to one another.

**Tasks:**

1. Model the data types for the artifacts ( `[A]` ) and the result of the analysis ( `[R]` ).
2. Design the following function:

```
// [R] is your type for the analysis result
//     (e.g., VALUABLE, HAZARDOUS, etc.)
// [A] is your type for an artifact.

/**
 * Performs a comparative analysis of an artifact
 * owned by a Rational Scavenger and a new artifact.
 * Returns the analysis of the newArtifact (the second
 * artifact) in relation to the ownedArtifact (the
 * first artifact) which can be valuable, hazardous, mundane,
 * incompatible, or unknown.
 */
[R] rationalScavengerAnalysis([A] ownedArtifact, [A] newArtifact)
```

3. Design the following functions to be used for testing. This is the testing interface so you do not need to add documentation to these functions:

```
// To interpret the result type [R]
boolean isValuable([R] result);
boolean isHazardous([R] result);
boolean isMundane([R] result);
boolean isIncompatible([R] result);
boolean isUnknown([R] result);

// To construct artifact values for testing
```

```
[A] makeStarChart(String dest, int risk, int sector, int system);
[A] makeEnergyCrystal(int power);
[A] makeInertRock(String color);
```

### Risk Taker Scavenger

You will design a function representing how a risk taking scavengers decides which artifact they prefer between an artifact they have and a new artifact. The analysis rules are as follows:

- If the new artifact is a **Star Chart**, this new star chart is always considered `VALUABLE` regardless of the owned artifact.
- When an owned **Star Chart** is compared to a new **Energy Crystal** or a new **Inert Rock**, the result is MUNDANE.
- When comparing two **Energy Crystals**, if the new crystal has a *higher* power level than the crystal they own, the new crystal is considered `VALUABLE` ; otherwise, it is `MUNDANE` .
- When comparing two **Inert Rocks**, if they are the same color, the new rock is considered `MUNDANE` . If their colors are different, a coin flip determines the outcome: there's a 50% chance the new rock is considered `VALUABLE` , and `INCOMPATIBLE` otherwise.
- When comparing **Energy Crystals** and **Inert Rock** in any order yields an `UNKNOWN` result.

**Tasks:**

1. Using the same data types from the Rational Scavenger, design the following function:

```
// [R] is your type for the analysis result
// (e.g., VALUABLE, HAZARDOUS, etc.)
// [A] is your type for an artifact.

/**
 * Performs a comparative analysis of an artifact
 * owned by a  Risk Taker Scavenger and a new artifact.
 * Returns whether the first artifact is valuable,
 * hazardous, mundane, incompatible, or unknown in
 * relation to the second.
 */
[R] riskTakerScavengerAnalysis([A] ownedArtifact, [A] newArtifact
```

## Part 2: The Scavenger Fleet

Galactic Scavengers are intrepid explorers. Each scavenger has a name, a single artifact in their cargo hold, and a unique **personal analysis protocol** for evaluating artifacts, which is a `BiFunction`. The functions `rationalScavengerAnalysis` and `riskTakerScavengerAnalysis` are examples of analysis protocols. We will model two distinct scenarios that use different analysis methods.

### Scenario 1: Exploring an Asteroid

When a scavenger finds an artifact on an asteroid, the outcome is determined by an analysis based on their analysis protocol from Part 1 for all scavengers.

1. Model the data types for the scavengers ( `[S]` ).
2. Design the following function:

```
// [S] is your type for a Scavenger.

/**
 * Computes the result of a scavenger finding an artifact
 * on an asteroid. This function must use the analysis protocol
 * for the scavenger to get the evaluation of the new artifact,
 * then apply the following rules to determine the final state
 * of the scavenger and their cargo:
 * - If the result is VALUABLE, the scavenger swaps artifacts.
 * - If MUNDANE, the scavenger ignores the new artifact.
 * - If HAZARDOUS, there is a 50% chance the ship's shields hold
 *   If they hold, the scavenger swaps. If they fail, the
 *   scavenger's current cargo is destroyed and replaced with
 *   a Inert Rock of color "dull grey".
 * - If INCOMPATIBLE or UNKNOWN, the scavenger ignores the
 *   new artifact.
 * The function returns a Pair of the scavenger's new state
 * and the artifact left behind.
 */
Pair<[S], [A]> exploreAsteroid([S] scavenger, [A] foundArtifact);
```

### Scenario 2: Trading at a Starport

When two scavengers meet to trade, the outcome is determined by their **analysis protocols** and a trade occurs **if and only if the trade is mutually beneficial.**

1. Design the following function:

```
    /**
     * Computes the result of two scavengers meeting to trade.
     * A trade occurs only if Scavenger A evaluates Scavenger
     * B's artifact as VALUABLE (using A's personal protocol),
     * AND Scavenger B evaluates Scavenger A's artifact as
     * VALUABLE (using B's personal protocol). The function
     * returns a Pair of the scavengers' final states, in their
     * original order.
     */
    Pair<[S], [S]> tradeAtStarport([S] scavengerA, [S] scavengerB);
```

**Testing Interface for Part 2:**

```
// To construct a scavenger with their personal protocol for testing
[S] makeScavenger(String name,
                  BiFunction<[A], [A], [R]> personalProtocol,
                  [A] initialCargo);

// To access a scavenger's properties
[A] getCargo([S] scavenger);
String getName([S] scavenger);
```

## Part 3: Simulating Encounters

Write a `main` function that simulates a Rational Scavenger's encounter based on a single line of input from the console. The program will parse the line, simulate the correct event using your functions from Parts 1 and 2, and print a description of the artifact the scavenger possesses *after* the encounter. Assume that all other scavengers found in trading posts are Risk Taker Scavengers.

### Understanding the Log Syntax

To complete this part, you must first understand the required syntax, which is broken into two parts.

#### 1. Syntax for describing an Artifact

This is the format representing an artifact. Your `describeArtifact` helper function must produce this format.

- **For a Star Chart:**
  `StarChart:destination;RISK=risk;SEC=sector;SYS=system`
  - **Example:** `"StarChart:Proxima Centauri;RISK=3;SEC=7;SYS=42"`
- **For an Energy Crystal:** `EnergyCrystal:POWER=power`

  - **Example:** `"EnergyCrystal:POWER=900"`
- **For an Inert Rock:** `InertRock:COLOR=color`
  - **Example:** `"InertRock:COLOR=obsidian"`

### 2. Syntax for a Log Entry

This is the full line of input your `main` function must parse. This line contains the type of encounter and two strings describing the current owned artifact and the other artifact using the syntax described above. Note that the delimiter is a pipe character `|` surrounded by single spaces.

- **For an Asteroid Encounter:**
  `ASTEROID | [ownedArtifact] | [foundArtifact]`
  - **Example:**
    `ASTEROID | InertRock:COLOR=quartz | StarChart:Proxima Centauri;RISK=3;SEC=7;SYS=42`
- **For a Trading Post Encounter:**
  `TRADING_POST | [ownedArtifact] | [otherScavengersArtifact]`
  - **Example:**
    `TRADING_POST | EnergyCrystal:POWER=500 | EnergyCrystal:POWER=900`

### Output Format

Since the log is written from the perspective of a Rational Scavenger, all asteroid encounters use the `rationalScavengerAnalysis` function from Part 1. Assume that all trades are done with Risk Taker Scavengers, that is, the `otherScavengersArtifact` is owned by a Risk Taker Scavenger. After simulating the event, your program must print a single line describing the final artifact held by the Rational Scavenger.

- **Format:** `FINAL CARGO: [final_artifact]`
- **Example Output:** `FINAL CARGO: EnergyCrystal:POWER=900`

### Tasks

1. Create the following function to return the artifact string representation

```
/**
 * Takes an artifact and returns a string of its details,
 * formatted exactly as required for the log entries.
 * - Star Chart: "StarChart:destination;RISK=risk;SEC=sector;SYS=
 * - Energy Crystal: "EnergyCrystal:POWER=power"
 * - Inert Rock: "InertRock:COLOR=color"
```

```
     */
    String describeArtifact([A] artifact);
```

2. Provide the following function to parse a Rational Scavenger Log:

```
    /**
     * parses a log string from a Rational Scavenger that takes
     * one of the following forms:
     * - ASTEROID | [ownedArtifact] | [foundArtifact]
     * - TRADING_POST | [ownedArtifact] | [otherScavengersArtifact]
     * Then it simulates the encounter and returns the
     * artifact the Rational Scavenger possesses *after*
     * the log entry
     */
    [A] parseRationalScavengerLog(String s);
```

3. Write a `main` function that asks for exactly one line input and parses that line as a Rational Scavenger Log entry. Then, simulate the correct event using your functions from Parts 1 and 2 and print a description of the artifact the scavenger possesses *after* the encounter. Assume that all other scavengers found in trading posts are Risk Taker Scavengers.

## Part 4

*In a file called `BouncingMarbles.java`*

Design a **world program** which simulates four different marbles moving around at a constant speed within a rectangular world 300 pixels wide and 500 pixels high. The world background color is white. All marbles are drawn as circles of radius 10 pixels. Each marble has its own colour different from each other's colour. The marble colours are `BLUE`, `RED`, `GREEN`, and `BLACK` for the first, second, third, and fourth marbles, respectively.

At any world program step, every marble has a position and a direction it is heading towards. The position of the marble is defined as the position of the pin of the marble (i.e., the center of the marble) within the background rectangle. The direction of the marble can be one of 8 different possibilities: (1) North, (2) South, (3) East and (4) West (known as cardinal directions); (5) NorthEast, (6) NorthWest, (7) SouthEast and (8) SouthWest (known as ordinal directions). Marbles move one pixel per step for the cardinal directions, and one pixel per direction per step in the case of ordinal directions. Thus, for example, if the current direction of a marble is North, then it moves one pixel North per step, and if the current direction is SouthWest, then it moves one pixel South and one pixel West per step.

If right at the beginning of a given world program step, the program detects that a marble has reached the boundary of the background rectangle, then the marble first bounces, i.e.,

its direction changes, and then it moves in the new direction within the same step. Which direction it changes to is a function of (1) the current's marble direction; and (2) the part of the boundary the marble has reached. We can split the boundary into 8 different parts: (1) top edge, (2) bottom edge, (3) left edge; (4) right edge; (5) top left corner; (6) top right corner; (7) bottom left corner, and (8) bottom right corner.

Thus, for example, if the marble reaches the top edge, we can distinguish among the following three scenarios:

1. Marble's current direction is North; its direction should change to South.
2. Marble's current direction is NorthEast; its direction should change to SouthEast.
3. Marble's current direction is NorthWest; its direction should change SouthWest.

For any other current direction, the Marble's direction should NOT change.

As another example, if the marble reaches the top right corner, we can distinguish among the following three scenarios:

1. Marble's current direction is North; its direction should change to South.
2. Marble's current direction is East; its direction should change to West.
3. Marble's current direction is NorthEast; its direction should change to SouthWest.
4. Marble's current direction is SouthEast; its direction should change to SouthWest.
5. Marble's current direction is NorthWest; its direction should change to SouthWest.

For any other current direction, the Marble's direction should NOT change.

For simplicity, the program should not handle in any way collisions among marbles. However, the user might be able to change the direction of the marbles with the mouse an keyboard. In particular, if the user hits the SPACE bar, the direction for every marble should randomly change to one of the cardinal directions. Besides, if the user left clicks the mouse, the direction for every marble should randomly change to one of the ordinal directions. Note that the position of the marble should not be altered as a consequence of these mouse and keyboard actions. Only their direction.

The initial position of (the centers of) the first, second, third and fourth marbles should be (X=75,Y=125), (X=225,Y=125), (X=75,Y=375), and (X=225,Y=375), respectively, and their initial direction randomly decided.

Follow the Design Recipe!

Provide the following testing interface:

```
// [W] should be replaced with whatever type you use to represent
//     the world state
// [D] should be replaced with whatever type you use to represent
//     the direction of a marble
// [M] should be replaced with whatever type you use to represent
```

```
//     a marble

// your stepping function
[W] step([W]);

// your drawing function
Image draw([W]);

// your mouse event handling function
[W] mouseEvent([W], MouseEvent);

// your key event handling function
[W] keyEvent([W], KeyEvent);

// returns the initial state of your world
[W] getInitialState();

// return the marbles in the world
[M] getMarble1([W]);
[M] getMarble2([W]);
[M] getMarble3([W]);
[M] getMarble4([W]);

// coordinates of the center of a marble
int getX([M]);
int getY([M]);

// direction of a marble
[D] getDirection([M]);

// to interpret the direction [D]
boolean isNorth([D]);
boolean isSouth([D]);
boolean isEast([D]);
boolean isWest([D]);
boolean isNorthEast([D]);
boolean isNorthWest([D]);
boolean isSouthEast([D]);
boolean isSouthWest([D]);
```

# Updates

| | |
|---|---|
| Sunday, 03 August 2025, 20:53 | Part 4. Added missing cases of changed directions for marble in corner example. |