- Don't forget to set your Eclipse workspace and working set.
- You must submit the JAR file, exported (with source code), from your Eclipse project.
- You must check your JAR file to make sure all the source files (.java files) are present. It can be opened with file compression programs such as 7-zip or Winrar.
- Failure to export properly will result in your work not getting marked.

To submit:

- Export your project to a JAR file, with source code.
- Name your JAR file ID_Week11_Q1.jar. For example, 6623110021_Week11_Q1.jar
- Submit the JAR file on MyCourseville.

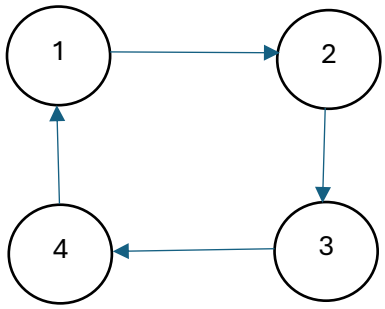
A Coordinate in a computer game's racetrack is represented by class **Point**.

A **Point** has the following values:

- value: the point's main ID.
- nextValue: ID of the next point.
- preValue: ID of the previous point.

In this question, we will represent a Point by a tuple (value, nextValue, preValue)

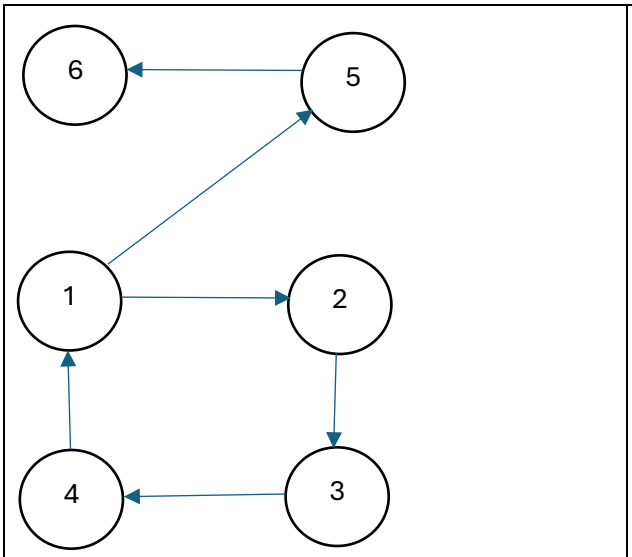
An example racetrack that's looks like:



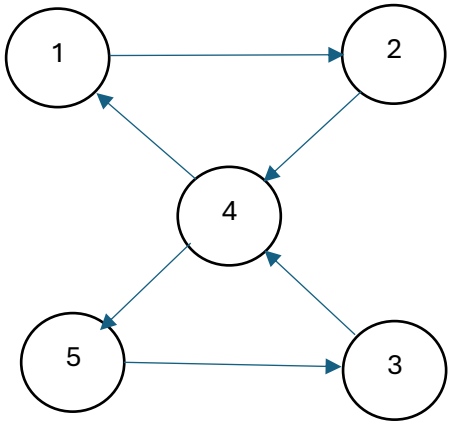
Will be represented by (1,2,4), (2,3,1), (3,4,2), (4,1,3). A node that does not have its next node will have its nextValue == -1. **A racetrack does not have to have a complete loop.**

A crossroad in a racetrack takes place where a Point with the same main value but different nextValue or different preValue exists (Just use this definition, don't think of any other possible scenarios).

The following racetracks have a crossroad (Points with same value but different nextValue or preValue will be stored as different data. Points with all the same value, nextValue and preValue won't be duplicated).



(1,2,4), (2,3,1), (3,4,2), (4,1,3), (1,5,4), (5,6,1), (6,-1,5)



(1,2,4), (2,4,1), (4,5,2), (4,5,3), (4,1,2), (4,1,3), (5,3,4), (3,4,5)

We are using a hash table to store Points as we build a racetrack.

A separate chaining hash table for storing Points, **SepChainingPoints2**, has been implemented. It stores an array of linkedlist for Points (**PointsLinkedList2**). Iterator and node were changed to **PointListIterator** and **PointListNode** to support Points in the hash table.

PointsLinkedList2 extends from **PointsLinkedList.** They have the same fields and methods, but you are to write an extra method for **PointsLinkedList2.**

SepChainingPoints2 extends from **SepChainingPoints.** They have the same fields and methods, but you are to write an extra method for **SepChainingPoints2.**

Your task is to implement one method in **PointsLinkedList2**, and one method in **SepChainingPoints2**.

In **PointsLinkedList2,** implement method:

- `public boolean pointValueExist(Point p) throws Exception`
  - This method returns **true** if there is a point in our list that has the same **value** as p, but different **nextValue** or different **preValue**.
  - Otherwise, this method returns **false**.

In **SepChainingPoints2**, implement method:

- `public boolean isCrossRoad(Point p) throws Exception`
  - This method stores p into our hash table.
  - It returns **true** if there is a Point in our hash table that has the same **value** as p, but different **nextValue** or different **preValue**. Otherwise, it returns **false**.
  - You can make use of method `pointValueExist(Point data)` of the hash table.
    - But make sure you finish writing `pointValueExist` of the **PointsLinkedList2** first.
    - (Note: hash function of Point **only uses value**, not **nextVale** and **preValue.**)

Scoring Criteria (5, will be scaled to equal to ant other question):

The files that will be tested will be **PointsLinkedList2.java** and **SepChainingPoints2.java**. **DO NOT** modify other files.

Junit

class TestPointsLinkedList

- testPointValueExist_NoCross() 1 mark
- testPointValueExist_Cross() 1 mark

class TestSepChainingPoints

If you do not use the given hash table, you get 0 point in this part.

- test_NoCross() 1 mark
- test_Cross1() 1 mark
- test_Cross2() 1 mark