COS 126	Princeton University	Spring 2026

Programming Exam

Before you begin. Read this page carefully. Do *not* start the exam or view any pages other than this cover page until instructed to do so.

Duration. You have 80 minutes to complete this exam.

Advice. *Review the entire exam before writing code. Implement the constructor and methods in the order given, one at a time, testing each one in* `main()` *before moving on.*

Submission. Submit your solutions on *TigerFile* using the link on the *Exams* page. You may submit multiple times, but only your last submission will be graded.

Check Submitted Files. You may click the *Check Submitted Files* button to receive *partial* feedback on your submission, up to 10 times. We will attempt to provide this feature during the exam, but you should not rely on it.

Grading. Your program will be graded *primarily* on correctness, but efficiency and clarity (including style) will also be considered. You will receive partial credit for a program that implements some of the required functionality. *You will receive a substantial penalty if your program does not compile.*

Allowed resources. During the exam, you may use only the following resources: the course textbook and companion booksite; lecture slides; course website; your course notes; your code from programming assignments or precepts; course *Ed*; course *codePost*, *Java visualizer*, and the *Oracle Javadoc*. Accessing any other website or resource is prohibited. For example, you may not use *Google* search, *Google Docs*, *Google Drive*, or *StackOverflow*. Generative AI tools, such as *ChatGPT*, and coding assistants are also prohibited.

No collaboration or communication. During the exam, collaboration and communication (including sharing files) are prohibited except with course staff. A staff member will be available to answer clarification questions.

No electronic devices or software. Software and computational or communication devices are prohibited except as needed to take this exam (such as a laptop, browser, and *IntelliJ*). For example, you must close all unnecessary virtual desktops, applications, and browser tabs; disable notifications; and *power off* all other electronic devices (including cell phones, tablets, calculators, cameras, smart watches, smart glasses, and earbuds). *You must use only the Princeton wireless network* eduroam*, not a mobile hotspot, VPN, Apple's Private Relay, or other network.*

Honor Code pledge. Write and sign the Honor Code pledge by typing the text below in the file `acknowledgments.txt`.

> *I pledge my honor that I will not violate the Honor Code during this examination.*

Electronically sign it by typing `/s/` followed by your name.

After the exam. Discussing or communicating the contents of this exam before solutions have been posted is a violation of the Honor Code. Accessing *TigerFile* is also prohibited.

Deliverables. For this programming exam, you will submit three files:

1. `Kingdom.java`, a data type representing a kingdom
2. `KingdomClient.java`, a client program
3. `acknowledgments.txt`, containing your Honor Code pledge

Project folder. Download the project folder from *TigerFile*. It contains:

- a template for `Kingdom.java`
- a template for `acknowledgments.txt`

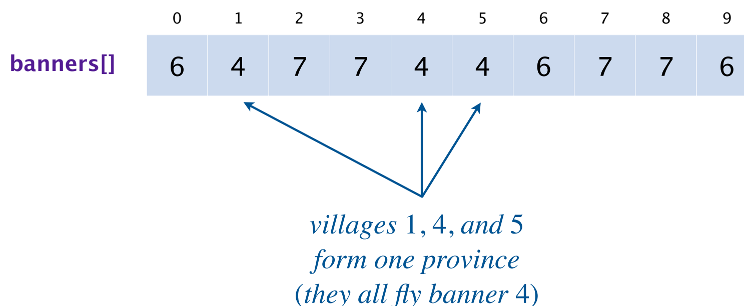
Library restrictions. On this exam, you may use classes defined *only* in `java.lang` (such as `Math` and `Double`) and our textbook libraries (such as `StdOut`, `In`, and `ST`). You may *not* use classes defined in `java.util` (such as `TreeMap` and `Arrays`).

Kingdom data type. A *kingdom* contains n *villages*, numbered 0 through $n-1$.

- Every village in the kingdom belongs to exactly one *province*; villages in the same province fly the same *banner*.
- Initially, each village is its own province, and village i flies banner i.
- Over time, leaders sign treaties that merge provinces. A treaty is identified by two villages p and q: it merges the province containing p into the province containing q. After the treaty, every village in the combined province flies the banner that village q flew before the treaty.

Kingdom representation. A natural representation is an integer array `banners[]` of length n, where `banners[i] = j` means that village i flies banner j. In this representation, each province consists of all villages whose `banners[]` entries have the same value.

For example, the following array represents a kingdom with 10 villages, partitioned into 3 provinces:



- One province (flying banner 4) consists of villages 1, 4, and 5.
- A second province (flying banner 6) consists of villages 0, 6, and 9.
- A third province (flying banner 7) consists of villages 2, 3, 7, and 8.

Kingdom API. Using the template file `Kingdom.java` provided in the project folder, write a data type that implements the following API:

`public class Kingdom`		
`public`	`Kingdom(int n)`	*creates a kingdom with $n > 0$ villages, numbered 0 through $n - 1$, each initially in its own province*
`public String`	`toString()`	*string representation (see below)*
`public int`	`bannerOf(int p)`	*the banner flown by village p*
`public boolean`	`inSameProvince(int p, int q)`	*returns* `true` *if villages p and q are in the same province, and* `false` *otherwise*
`public void`	`signTreaty(int p, int q)`	*merges the province containing p into the province containing q*
`public int`	`provinceCount()`	*number of provinces*
`public int`	`largestProvinceSize()`	*number of villages in the largest province*

Kingdom API details. Here is some additional information about the required behavior:

- The `toString()` method returns a string representing the banners of the n villages, with the banners separated by commas and spaces, and enclosed in square brackets. For example, the kingdom illustrated on the facing page has string representation

  ```
  [6, 4, 7, 7, 4, 4, 6, 7, 7, 6]
  ```

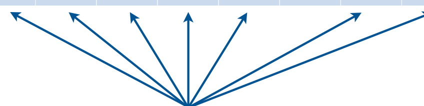
- The `signTreaty(p, q)` method merges the province containing p into the province containing q. After the treaty, every village in the combined province flies the banner that village q flew before the treaty. For example, here is the effect of calling `signTreaty(3, 1)`:

before treaty

0	1	2	3	4	5	6	7	8	9
6	4	7	7	4	4	6	7	7	6

after signTreaty(3, 1)

0	1	2	3	4	5	6	7	8	9
6	4	4	4	4	4	6	4	4	6



every village in combined province now flies banner 4

- *Corner cases.*
  - If p and q are in the same province before calling `signTreaty(p, q)`, then that call has no effect.
  - If any argument to `signTreaty()`, `bannerOf()`, or `inSameProvince()` is outside the range 0 through $n - 1$, throw an `IllegalArgumentException`.
  - You may assume the argument n to the constructor is positive.
- *Unit testing.* The `main()` method must directly call every instance method. You may use (or extend) the following code, which is provided in the `Kingdom.java` template.

```
/***************************************************************************
 *  The expected output of this unit test is:
 *
 *  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
 *  [6, 4, 7, 7, 4, 4, 6, 7, 7, 6]
 *  bannerOf(2)           = 7
 *  bannerOf(5)           = 4
 *  inSameProvince(2, 5)  = false
 *  provinceCount()       = 3
 *  largestProvinceSize() = 4
 *  [6, 4, 4, 4, 4, 4, 6, 4, 4, 6]
 ***************************************************************************/
public static void main(String[] args) {

    // create a kingdom with 10 villages and call all instance methods
    Kingdom kingdom = new Kingdom(10);
    StdOut.println(kingdom);
    kingdom.signTreaty(2, 3);
    kingdom.signTreaty(8, 7);
    kingdom.signTreaty(3, 7);
    kingdom.signTreaty(0, 9);
    kingdom.signTreaty(5, 4);
    kingdom.signTreaty(0, 6);
    kingdom.signTreaty(1, 4);
    StdOut.println(kingdom);
    StdOut.println("bannerOf(2)           = " + kingdom.bannerOf(2));
    StdOut.println("bannerOf(5)           = " + kingdom.bannerOf(5));
    StdOut.println("inSameProvince(2, 5)  = " + kingdom.inSameProvince(2, 5));
    StdOut.println("provinceCount()       = " + kingdom.provinceCount());
    StdOut.println("largestProvinceSize() = " + kingdom.largestProvinceSize());
    kingdom.signTreaty(3, 1);
    StdOut.println(kingdom);
}
```

Client program. Write a client program `KingdomClient.java` that takes the name of an input file as a command-line argument, creates a `Kingdom` object, processes the treaties in the file, and prints the resulting kingdom and its provinces in the format specified.

- **Input.** The input file begins with the number of villages n. The remainder of the file consists of zero or more treaties, one per line. Each treaty is given by two integers (between 0 and $n-1$) specifying the villages involved. (Assume all input files conform to this format.)
- **Output.** On the first line, print the string representation of the resulting kingdom. Then, print one province per line. Each line should contain the province's banner, followed by a colon, followed by the villages in that province in increasing order, separated by whitespace. (You may output the provinces in arbitrary order, not necessarily sorted by banner.)

Here is an input file `input10.txt` and sample execution:

```
~/Desktop/s26-pe> more input10.txt
10      <-- number of villages n
2 3
8 7
3 7
0 9
5 4     <-- a treaty (p = 5, q = 4)
0 6
1 4
                                                   filename
~/Desktop/s26-pe> java-introcs KingdomClient input10.txt
[6, 4, 7, 7, 4, 4, 6, 7, 7, 6]   <-- string representation of resulting kingdom
4: 1 4 5
6: 0 6 9   <-- villages in one province (flying banner 6)
7: 2 3 7 8
   ^ increasing order
```

Performance requirements. No constructor, instance method, or client should take more than $\Theta(n^2)$ time, where n is the number of villages.

Grading. This programming exam is worth a total of 50 points. Here is the breakdown:

part	*points*
constructor	6
`toString()`	6
`bannerOf()`	4
`inSameProvince()`	4

part	*points*
`signTreaty()`	6
`provinceCount()`	6
`largestProvinceSize()`	6
`KingdomClient`	12

You may earn full credit for `KingdomClient.java` even if `Kingdom.java` is not fully functional.