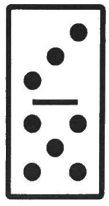
CH III Domino Constructorpalooza
COSC 2329 Component-Based Programming
Assigned: Monday, March 5, 2018
Due: Tuesday, March 20, 2018 6:00 PM
Late Deadline: No lates accepted

Anatomy of a Domino



Every domino has two ends containing *pips*. The domino above has two ends, one end has 3 *pips* and the other end has 5 *pips*. In classic dominoes, the minimum pip count for an end is 0 and the maximum pip count for an end is 6. So, there are 28 = 7 + 6 + 5 + 4 + 3 + 2 + 1 dominoes in the classic set. We consider the "[3:5]" domino above to be identical to this one:

That is, there are not the concepts of *top* or *bottom* of the domino. In addition, "rotating" a domino doesn't change it.

Domino Interface

```
package dominoes;

public interface Domino
{
	public static final int MINIMUM_PIP_COUNT = 0;
	public static final int MAXIMUM_PIP_COUNT = 6;
	//part of post: MINIMUM_PIP_COUNT <= rv <= MAXIMUM_PIP_COUNT
	//part of post: getLowPipCount() <= rv
	public int getHighPipCount();
	//part of post: MINIMUM_PIP_COUNT <= rv <= MAXIMUM_PIP_COUNT
	//part of post: rv <= getHighPipCount()
	public int getLowPipCount();
}
```

Domino Concept

A (non-null) `Domino` instance provides two services, it can:

- tell the client (aka, "caller") what its high pip count is
- tell the client (aka, "caller") what its low pip count is

Note that the `Domino` instance:

- receives its high and low pip counts at "time of manufacture" (i.e., construction),
- is not mutable; there is not a method in the interface that causes a change to what the instance would return in response to `getHighPipCount()` and `getLowPipCount()`.

Constructors

From the client's perspective, the service that a constructor offers clients is the ability to trade a specification (in some agreed-upon client-facing representation) for an instance of the particular class in which the constructor resides. So, the constructor constructs (i.e., manufactures, fabricates) instances "out of thin air" in response to a request (with an implicit or explicit specification) from the client.

From the Impl author's perspective, the job of the constructor is twofold:

1. interpret the client's manufacturing request (through the lens of the constructor/client-facing representation) to determine the desired instance
2. encode the desired instance (using the class' internal representation) in the class' instance variables.

This assignment requires you to figure out and be able to translate between eight distinct representations:

- `int highPipCount, int lowPipCount`
- `String highLowString`
- `int[] sumDifference`
- `Set<Integer> highLowSet`
- `String sumDifferenceString`
- `int lowPlus8TimesHigh`
- `String lowDifferenceString`
- `List<Integer> highSum`

The specifics of these representations will be discussed in class. Take special note that it is impossible to complete this assignment with certainty unless clarifying questions are asked. This is intentional. Being able to ask questions that distinguish between different representations is a necessary skill for computer programming - there are representations **ALL OVER THE PLACE!**

The good news is that this skill can be acquired through experience.

DominoHighLowImpl

```
package dominoes;

...

public class DominoHighLowImpl_Skeleton implements Domino
{
	private int highPipCount;
	private int lowPipCount;

	public DominoHighLowImpl_Skeleton (int highPipCount, int lowPipCount)
	{
		throw new RuntimeException("NOT IMPLEMENTED!");
	}

	public static final char HIGH_LOW_STRING_SEPARATOR = ':';

	public static boolean isHighLowString(String str)
	{
		throw new RuntimeException("NOT IMPLEMENTED!");
	}

	public DominoHighLowImpl_Skeleton(String highLowString)
	{
		throw new RuntimeException("NOT IMPLEMENTED!");
	}

	public static final int INDEX_OF_SUM = 0;
	public static final int INDEX_OF_DIFFERENCE = 1;
	//part of pre: sumDifference.length == 2
	//part of pre: sumDifference[INDEX_OF_SUM] >=
	//                  sumDifference[INDEX_OF_DIFFERENCE]
	public DominoHighLowImpl_Skeleton(int[] sumDifference)
	{
		throw new RuntimeException("NOT IMPLEMENTED!");
	}

	//part of pre: 1<= highLowSet.size() <= 2
	//part of pre: ! highLowSet.contains(null)
	public DominoHighLowImpl_Skeleton(Set<Integer> highLowSet)
	{
		throw new RuntimeException("NOT IMPLEMENTED!");
	}

	...
}
```

DominoHighLowSetImpl

```
package dominoes;
...
public class DominoHighLowSetImpl_Skeleton implements Domino
{
	private Set<Integer> highLowSet;

	public DominoHighLowSetImpl_Skeleton(int highPipCount, int lowPipCount)
	{
		throw new RuntimeException("NOT IMPLEMENTED!");
	}

	public static final char SUM_DIFFERENCE_DELIMITER = ',';
	public static boolean isSumDifferenceString(String str)
	{
		throw new RuntimeException("NOT IMPLEMENTED!");
	}

	public DominoHighLowSetImpl_Skeleton(String sumDifferenceString)
	{
		throw new RuntimeException("NOT IMPLEMENTED!");
	}

	public static boolean isLowPlus8TimesHighInteger(int k)
	{
		throw new RuntimeException("NOT IMPLEMENTED!");
	}

	public DominoHighLowSetImpl_Skeleton(int lowPlus8TimesHigh)
	{
		throw new RuntimeException("NOT IMPLEMENTED!");
	}
}
```

DominoLowDifferenceStringImpl

```
package dominoes;
...
public class DominoLowDifferenceStringImpl_Skeleton implements Domino
{
        private String lowDifferenceString;
        private static final char LOW_DIFFERENCE_DELIMITER = '*';

        //pre: left to student
        //post: left to student
        public DominoLowDifferenceStringImpl_Skeleton(int lowPlus$TimesHigh)
        {
                throw new RuntimeException("NOT IMPLEMENTED!");
        }


        public static final int INDEX_OF_HIGH = 0;
        public static final int INDEX_OF_SUM = 1;

        //pre: left to student
        //post: left to student
        public DominoLowDifferenceStringImpl_Skeleton(
                                        List<Integer> highSum)
        {
                throw new RuntimeException("NOT IMPLEMENTED!");
        }
}
```

Assignment

Your assignment is to create four properly documented constructs in Java:

- a Domino interface (see below) [**You will not be submitting this file – I will use my own**]
- a `DominoHighLowImpl_`*`LastName`* class:
  (e.g., `DominoHighLowImpl_Kart`), which implements the `Domino` interface and includes the four mandatory constructors (see the "`DominoHighLowImpl`" section above)
- a DominoHighLowSetImpl_*LastName* class:
  (e.g., `DominoHighLowSetImpl_Kart`), which implements the `Domino` interface and includes the three mandatory constructors (see the "`DominoHighLowSetImpl`" section above)
- a DominoLowDifferenceStringImpl_*LastName* class:
  (e.g., `DominoLowDifferenceStringImpl_Kart`), which implements the `Domino` interface and includes the two mandatory constructors (see the "`DominoLowDifferenceStringImpl`" section above)

Deliverables

- A .zip file uploaded to Canvas that contains the following files
  (Look for "Domino" assignment or similar):
  - `DominoHighLowImpl_`*`LastName`*`.java` (e.g., `DominoHighLowImpl_Kart.java`)
  - DominoHighLowSetImpl_*LastName*`.java` (e.g., `DominoHighLowSetImpl_Kart.java`)
  - `DominoLowDifferenceStringImpl_`*`LastName`*`.java` (e.g., `DominoLowDifferenceStringImpl_Kart.java`)
  - Any supporting Utils/classes/interfaces that you created (note that the filename suffix on these files must be *_LastName*)
- **Printouts of your files, stapled together (bring to the class following the deadline)**

Rules

- My test cases do not change based on your submission.
- I will not violate the preconditions on my interface in my test cases.
- **USE THE PACKAGE 'dominoes' for all of your files!**
- Use the Eclipse IDE
- Ensure that I, with only modest effort, can understand your code
- Ensure that the code is properly documented
- Ensure that the code is properly formatted
- **Test your code!** (What test cases can you think of?)
  - What are the "middle-of-the-road" (i.e., "vanilla") test cases?
  - What are the "corner" (i.e., "extreme") test cases?
- **Test your code some more!** (What other test cases can you think of?)
- Code that doesn't compile will not pass any tests and receive a score of 0
- **Ensure that your files follow the naming convention under Deliverables**
- **WARNING: This specification may be misleading or incomplete! Part of the assignment is to read the assignment early, think about it, and ask any clarifying questions!**