Package Collation

Enum Class LocaleCategory

java.lang.Object
    java.lang.Enum<LocaleCategory>
        Collation.LocaleCategory

All Implemented Interfaces:

Serializable, Comparable<LocaleCategory>, Constable

```
public enum LocaleCategory
extends Enum<LocaleCategory>
```

The purposes that locales serve are grouped into categories, so that a user or a program can choose the locale for each category independently. The following is all the available categories; each name is both an environment variable that a user can set, and a macro name that you can use as the first argument to `Collation.setLocale`.

Nested Class Summary

Nested classes/interfaces inherited from class java.lang.Enum

`Enum.EnumDesc<E extends Enum<E>>`

Enum Constant Summary

Enum Constants

Enum Constant	Description
LANG	
LC_ALL	
LC_COLLATE	
LC_CTYPE	
LC_MESSAGES	
LC_MONETARY	

LC_NUMERIC
LC_TIME

Method Summary

All Methods | Static Methods | Concrete Methods

Modifier and Type	Method	Description
static LocaleCategory	valueOf(String name)	Returns the enum constant of this class with the specified name.
static LocaleCategory[]	values()	Returns an array containing the constants of this enum class, in the order they are declared.

Methods inherited from class java.lang.Enum

clone, compareTo, describeConstable, equals, finalize, getDeclaringClass, hashCode, name, ordinal, toString, valueOf

Methods inherited from class java.lang.Object

getClass, notify, notifyAll, wait, wait, wait

Enum Constant Details

LC_COLLATE

```
public static final LocaleCategory LC_COLLATE
```

LC_CTYPE

```
public static final LocaleCategory LC_CTYPE
```

LC_MONETARY

```
public static final LocaleCategory LC_MONETARY
```

LC_NUMERIC

```
public static final LocaleCategory LC_NUMERIC
```

LC_TIME

```
public static final LocaleCategory LC_TIME
```

LC_MESSAGES

```
public static final LocaleCategory LC_MESSAGES
```

LC_ALL

```
public static final LocaleCategory LC_ALL
```

LANG

```
public static final LocaleCategory LANG
```

Method Details

values

```
public static LocaleCategory[] values()
```

Returns an array containing the constants of this enum class, in the order they are declared.

Returns:

an array containing the constants of this enum class, in the order they are declared

valueOf

```
public static LocaleCategory valueOf(String name)
```

Returns the enum constant of this class with the specified name. The string must match *exactly* an identifier used to declare an enum constant in this class. (Extraneous whitespace characters are not permitted.)

Parameters:

`name` - the name of the enum constant to be returned.

Returns:

the enum constant with the specified name

Throws:

`IllegalArgumentException` - if this enum class has no constant with the specified name

`NullPointerException` - if the argument is null

Package Collation

Class Collation

java.lang.Object
    Collation.Collation

```
public class Collation
extends Object
```

The `Collation` class provides locale-specific string comparison and transformation capabilities.

This class includes methods for:

- Setting and retrieving the locale for specific `LocaleCategory` categories.
- Lexicographically comparing strings based on the collation rules of the current locale.
- Transforming a specified number of characters in a string according to locale-specific collation settings.

Example Usage:

```
Collation collation = new Collation();
boolean success = collation.setLocale(LocaleCategory.LC_ALL, "zh_CN.GB2312");
if (success) {
     System.out.println(collation.compareStringWithCollation("你好", "你好世界")); // expect a negative number
     System.out.println(collation.transformStringWithCollation("你好世界", 2)); // expect to be "你好"
}
```

Constructor Summary

Constructors

Constructor	Description
`Collation()`	Collation Constructor, initialize the class, it does: 1.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
`int`	`compareStringWithCollation(String s1, String s2)`	Lexicographically compare two strings using the collating sequence of the current locale for collation.
`String`	`getLocale(LocaleCategory localeCategory)`	Get current locale associated with the `localeCategory`
`boolean`	`setLocale(LocaleCategory localeCategory, String local)`	The function sets the current locale for the `localeCategory` to be `locale`.
`String`	`transformStringWithCollation(String from, int size)`	The function transforms a specified number of characters, given by `size`, from the string `from` using a collation transformation based on the currently selected locale, and returns the transformed string.

Methods inherited from class java.lang.Object

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

Constructor Details

Collation

```
public Collation()
```

Collation Constructor, initialize the class, it does: 1. search current environment's locale 2. set the `LocaleCategory` and local inherited from the environment

Method Details

setLocale

```
public boolean setLocale(LocaleCategory localeCategory,
                         String local)
```

The function sets the current locale for the `localeCategory` to be `locale`.

The method is not safe to use in multi-thread programs without additional synchronization.

Parameters:

`localeCategory` - If category is `LC_ALL`, this specifies the locale for all purposes. The other possible values of category specify a single purpose (see `LocaleCategory`).

`local` - the local name represented by String. The command `locale -a` prints all the local names supported by the current system. This argument is expected to be one of these names.

Returns:

true if the specified local name is valid, false otherwise. The current locale will not be unchanged if the local name is invalid

getLocale

```
public String getLocale(LocaleCategory localeCategory)
```

Get current locale associated with the `localeCategory`

Returns:

a string that is the name of the locale currently selected for `LocaleCategory`

compareStringWithCollation

```
public int compareStringWithCollation(String s1,
                                      String s2)
```

Lexicographically compare two strings using the collating sequence of the current locale for collation. The current locale can be retrieved by `getLocale(LocaleCategory.LC_COLLATE)`.

The method is safe to use in multi-thread programs without additional synchronization.

Parameters:

`s1` - the first string to be compared.

`s2` - the second string to be compared.

Returns:

a positive integer if `str1` object lexicographically precedes the `str2`. The result is a negative integer if `str1` is lexicographically smaller than `str2`. The result is zero if `str1` and `str2` are equal.

transformStringWithCollation

```
public String transformStringWithCollation(String from,
                                           int size)
                                    throws IllegalArgumentException
```

The function transforms a specified number of characters, given by `size`, from the string `from` using a collation transformation based on the currently selected locale, and returns the transformed string. Up to size bytes (including a terminating null byte) are stored.

The transformed string may be longer than the original string, and it may also be shorter.

The method is safe to use in multi-thread programs without additional synchronization.

Parameters:

`from` - the String to be transformed from

`size` - the number of characters in `from` to transform

Returns:

a string `size` bytes

Throws:

`IllegalArgumentException` - if size ≤ 0, or if the `size` is larger than the number of characters in the String `from`. Note: if `from` contains Unicode characters, its number of characters is `from.codePointCount(0, from.length());`, else the number of characters is `from.length()`

Rational of Collation Function Translation
The design adapts C collation functions (`strcoll`, `strxfrm`, `wcscoll`, `wcsxfrm`) into Java's object-oriented and Unicode-compatible environment, at the same time improvements for clarity, and alignment with Java's conventions.

Design Choices and Naming

1. Collation Class:
  - `Collation`: The `Collation` class name reflects its primary purpose of handling locale-specific collation.
  - `setLocale` and `getLocale`: In C, `setlocale` serves both to set and retrieve the locale, which is unexpected and can surprise the user. By splitting this functionality into two distinct methods, the design enhances readability.
  - `compareStringWithCollation`: This method corresponds to C's `strcoll` and `wcscoll`, which compare two strings based on locale rules. In C, `strcoll` and `wcscoll` are necessary because `char` is single-byte and `wchar_t` supports multi-byte or wide characters. However, Java's `String` inherently supports Unicode, so we combine these two functions into one. The naming was also adjusted to improve clarity, avoiding abbreviations like 'w' and 'coll', and explicitly mentioning "compare."
  - `transformStringWithCollation`: This method adapts `strxfrm` and `wcsxfrm`, applying locale-based transformations to a portion of a string. In C, separate functions for `char` and `wchar_t` were necessary, but Java's `String` supports Unicode, so both functions are also combined here.
In the Java translation, memory overlap checks were eliminated because Java's immutable String and memory management prevent such issues. In `C`, the `size` argument is unsigned, but java doesn't have unsigned. So added argument check for `size` to throw exceptions for negative input.

2. Enum `LocaleCategory`:
  - The `LocaleCategory` enum represents specific locale categories, such as `LC_COLLATE` and `LC_CTYPE`. It mirrors the original design in C (using the same enum values), making it easier for users familiar with C to transition to this Java API.

Translation Design Principles Used

1. Java-Specific Adjustments:
  - Java's `String` class supports Unicode, which eliminates the need for separate functions for narrow (`char`) and wide (`wchar_t`) characters as in C.

2. Consistency with Java Conventions:
  - The API follows Java naming conventions, such as camelCase for methods, and adopts an object-oriented design.

3. Translation with Improvements:
  - The API translates all C functions and closely follows the original function names unless the naming was unclear or could be misleading (such as `setlocale`, `strcoll`, `wcscoll`, `strxfrm`, `wcsxfrm`).

Class INIParser

java.lang.Object
    INIParser

```
public class INIParser
extends Object
```

Parser for INI files. This class allows loading, modifying, and querying configuration settings stored in INI format.

INI files store configuration data in a structured format with sections and key-value pairs. This API supports retrieving values as strings, integers, doubles, and booleans, as well as setting or unsetting entries and accessing sections directly.

This API is translated from the iniparser API found here: iniparser GitHub

Thread Safety: The `INIParser` class is not thread-safe. Access to the dictionary (i.e., loading, modifying, and querying) should be synchronized externally if used in a concurrent environment.

Example Usage:

```
INIParser parser = new INIParser();

// Load INI file
try {
    parser.load("config.ini");
} catch (IOException e) {
    System.err.println("Failed to load INI file.");
}

// Retrieve values
String value = parser.getString("section:key", "default");
int intValue = parser.getInt("section:key", -1);
boolean boolValue = parser.getBoolean("section:key", false);

// Set and remove values
parser.setEntry("section:key", "newValue");
parser.unsetEntry("section:key");

// Dump contents to System.out
parser.dump(System.out);
```

Constructor Summary

Constructors

Constructor	Description
`INIParser()`	Constructs an empty INIParser instance with an empty configuration dictionary.

Method Summary

All Methods | **Static Methods** | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
void	`dump(PrintStream out)`	Dumps all entries in the dictionary to the specified PrintStream.
boolean	`findEntry(String entry)`	Checks if a specific entry is present within the dictionary.
boolean	`getBoolean(String key, boolean defaultValue)`	Retrieves the boolean value associated with the specified key.
int	`getInt(String key, int defaultValue)`	Retrieves the integer value associated with the specified key.
int	`getSectionCount()`	Counts the number of unique sections within the dictionary.
String	`getSectionName(int index)`	Retrieves the name of a section at a specified index.
String	`getString(String key, String defaultValue)`	Retrieves the string value associated with the specified key.
Map<String,String>	`load(String fileName)`	Loads the contents of an INI file, populating the dictionary with parsed entries.
Map<String,String>	`loadFromReader (BufferedReader reader)`	Loads the contents of an INI file from a BufferedReader, populating the dictionary.
int	`setEntry(String entry, String value)`	Sets or updates an entry in the dictionary with the

		specified value.
static void	**setErrorCallback**(**PrintStream** errCallback)	Sets a custom error output stream for reporting syntax errors or missing entries.
void	**unsetEntry**(**String** entry)	Removes an entry from the dictionary if it exists.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Details

INIParser

```
public INIParser()
```

Constructs an empty INIParser instance with an empty configuration dictionary.

Method Details

setErrorCallback

```
public static void setErrorCallback(PrintStream errCallback)
```

Sets a custom error output stream for reporting syntax errors or missing entries.

Parameters:

`errCallback` - a PrintStream for error output, such as `System.err` or a custom logging stream. If null, defaults to `System.err`.

load

```
public Map<String,String> load(String fileName)
                        throws IOException
```

Loads the contents of an INI file, populating the dictionary with parsed entries.

Parameters:

`fileName` - the name of the INI file to parse

Returns:

the populated dictionary of parsed entries

Throws:

IOException - if the file cannot be read

Example Usage:

```
INIParser parser = new INIParser();
try {
    parser.load("config.ini");
} catch (IOException e) {
    e.printStackTrace();
}
```

loadFromReader

```
public Map<String,String> loadFromReader(BufferedReader reader)
                                  throws IOException
```

Loads the contents of an INI file from a BufferedReader, populating the dictionary.

Parameters:

`reader` - BufferedReader providing the INI file contents

Returns:

the populated dictionary of parsed entries

Throws:

IOException - if an error occurs while reading

getString

```
public String getString(String key,
                        String defaultValue)
```

Retrieves the string value associated with the specified key.

Parameters:

`key` - the key to retrieve the value for

`defaultValue` - the default value if the key is not found

Returns:

the value associated with the key, or `defaultValue` if the key is absent

getInt

```
public int getInt(String key,
                  int defaultValue)
```

Retrieves the integer value associated with the specified key.

Parameters:

`key` - the key to retrieve the value for

`defaultValue` - the default integer value if the key is not found or cannot be parsed as an integer

Returns:

the integer value, or `defaultValue` if the key is absent or invalid

getBoolean

```
public boolean getBoolean(String key,
                          boolean defaultValue)
```

Retrieves the boolean value associated with the specified key.

Parameters:

`key` - the key to retrieve the value for

`defaultValue` - the default boolean value if the key is not found or cannot be parsed as a boolean

Returns:

the boolean value, or `defaultValue` if the key is absent or invalid

getSectionCount

```
public int getSectionCount()
```

Counts the number of unique sections within the dictionary.

Returns:

the total number of sections found in the dictionary

getSectionName

```
public String getSectionName(int index)
```

Retrieves the name of a section at a specified index.

Parameters:

`index` - the index of the section name to retrieve

Returns:

the name of the section at the specified index, or `null` if the index is out of bounds

findEntry

```
public boolean findEntry(String entry)
```

Checks if a specific entry is present within the dictionary.

Parameters:

`entry` - the entry to search for

Returns:

`true` if the entry exists, `false` otherwise

setEntry

```
public int setEntry(String entry,
                    String value)
```

Sets or updates an entry in the dictionary with the specified value.

Parameters:

`entry` - the entry key

`value` - the value to associate with the entry

Returns:

0 if set successfully, or -1 if the entry is null or empty

unsetEntry

```
public void unsetEntry(String entry)
```

Removes an entry from the dictionary if it exists.

Parameters:

`entry` - the entry to remove

dump

```
public void dump(PrintStream out)
```

Dumps all entries in the dictionary to the specified PrintStream.

Parameters:

`out` - the PrintStream to write dictionary contents to, e.g., `System.out`

INIParser Translation Rationale

Platform Conventions and Principles

Java is a high-level, object-oriented language with robust standard libraries. Unlike C, which relies on manual memory management, Java's runtime manages memory with garbage collection, reducing the need for explicit allocation and freeing of resources. Java also provides a rich set of data structures (like Map as I used) that abstract away the need for custom implementations, unlike C, where developers often implement data structures like dictionaries manually. These conventions guided me in translating INIParser from C to Java.

Design Decisions and Alternatives

1. Data Structure Choice: HashMap over Custom Dictionary

Original C Design: The original INIParser in C relied on dictionary.c and dictionary.h for managing key-value pairs, since C lacks a standard library for associative arrays or hash maps.

Java Approach: In Java, HashMap<String, String> replaced the custom dictionary structure. Java's HashMap is optimized for key-value storage and provides O(1) access time, which is ideal for our needs in storing and retrieving INI file entries.

- Strengths:
    - HashMap is managed by Java's garbage collector, eliminating manual memory allocation and deallocation.
    - The built-in implementation is optimized for speed, readability, and maintainability.
- Weaknesses:
    - With HashMap, we have less control over low-level memory operations compared to the C implementation, though it might not be necessary.

Decision: The HashMap was chosen because it simplifies the design, reduces code complexity, and aligns with Java's built-in collection standards. This allowed us to eliminate dictionary.c and dictionary.h.

2. Error Handling: Exceptions over Return Codes

Original C Design: In C, functions typically return integer codes (e.g., -1 for errors).

Java Approach: Java's standard for error handling uses exceptions, which provide a more explicit way to handle and report errors.

- Strengths:
    - Exceptions make it clear where errors occur and provide more informative messages.
    - This simplifies debugging and error management.
- Weaknesses:
    - Those who never used Java may have a small learning curve.

Decision: Exceptions were chosen to align with Java's conventions, improve readability, and leverage structured error handling. For instance, IOException is thrown if the file fails to load, rather than returning an error code.

3. Simplified Method Set Using Java's Built-in Methods

Original C Design: C required many helper functions for tasks like string manipulation, case conversion, and error handling. These include strlwc() for lowercase conversion, xstrdup() for string duplication, and etc.

Java Approach: Built-in methods are provided for handling strings and key-value pairs. Java automatically supports string case conversion (e.g. toLowerCase) and other helper functions.

- Strengths:
    - By removing redundant methods, the API is easier to read and maintain.
    - Java's built-in methods are optimized and tested, which are reliable.
- Weaknesses:
    - Same as before, we may lose some customizable low-level controls.

Decision: I removed unnecessary helper methods, relying on Java's standard library for similar functionality. For instance, strlwc() was replaced with toLowerCase(), and functions to maintain the heap are no longer needed, either.

4. Boolean Parsing: Flexibility in Parsing "True" and "False" Values

Original C Design: INIParser in C used iniparser_getboolean() to interpret boolean values flexibly, supporting values like "yes", "1", "true", etc.

Java Approach: Use methods like getBoolean().

- Strengths:
    - It's easier to develop and the code looks cleaner.
- Weaknesses:
    - Flexible parsing could lead to unexpected results if values are non-standard.

Decision: I retained flexible boolean parsing to preserve compatibility with INI file formats, ensuring a seamless transition from the original API to Java.

5. Simplified Initialization and Section/Key Parsing

Original C Design: The original API required iniparser_load and various dictionary_set calls to initialize and populate the dictionary.

Java Approach: Java's HashMap and streamlined load method simplify the initialization process. I also combined section and key parsing into a single parseKeyValue method to reduce complexity.

- Strengths:
    - It's easier to develop.
    - Readability: Java's structure allows more readable initialization and parsing logic.
- Weaknesses:
    - Loss of Granularity: With combined methods, developers lose some granularity for customizing sections and key parsing.
- Decision: I streamlined initialization and parsing for simplicity, which is beneficial given Java's high-level abstractions and typical usage patterns.

Summary

The translation from C to Java of the INIParser API focused on maintaining core functionality while leveraging Java's strengths in object-oriented design, memory management, and error handling. By using Java's standard library, I eliminated many custom C functions, making the API more approachable and maintainable. Key changes included:

These design decisions, grounded in Java's platform conventions, make the INIParser API more intuitive for Java developers, while preserving the essential functionality of the original C API. This approach provides a robust, readable, and maintainable API for working with INI files in Java.

Work History

1. Initial Planning and API Selection

- **Team Discussion**: As a team, we convened to discuss potential APIs for translation, considering the assignment requirements and caveats.
- **API Selection**: After reviewing the options, we agreed on two API translation tasks: (1) Locale-sensitive string collation and (2) DOS/Windows-style .ini file parsing.
- **Platform Selection**: After discussing our areas of expertise, we chose Java for translation.

2. API Translation Drafting

- **Translation Process**: Using the selected platform, we started drafting each API's translation, adhering to platform-specific conventions for clarity and usability.

3. Documentation and Sample Code

- **Documentation Creation**: We prepared clear, concise documentation for each API
- **Client Code Examples**: Developed sample client code that illustrates the usage of each API to validate functionality and demonstrate ease of use.

Contributions:

For step 2 and 3, we divided the work equally among all team members, with each person contributing to both the Locale-Sensitive String Collation and INI File Parsing translations.

4. Design Rationale and Revision

- **Design Rationale**: Drafted a summary of design decisions, explaining how platform conventions influenced the translation and decisions made to enhance usability and performance.
- **Revision Process**: After completing the initial draft, we reviewed and revised the APIs and the documentations based on internal feedback

Contributions:

Peitong: Wrote the design rationale for the Locale-sensitive string collation translation

Tianyin: Wrote the design rationale for the .ini file parsing translation

Bohan: Revise the APIs and the documentations

```java
import java.math.BigInteger;

/**
 * Represents an arbitrary-precision rational number (fraction) with a numerator and
denominator.
 * Provides functionality for basic arithmetic operations, comparison, and formatting.
 */
public class Rational {

    private final BigInteger numerator;
    private final BigInteger denominator;

    /**
     * Constructs a Rational number with the specified numerator and denominator.
     * The fraction is automatically simplified to its lowest terms.
     *
     * @param numerator the numerator of the rational number
     * @param denominator the denominator of the rational number; must not be zero
     * @throws IllegalArgumentException if the denominator is zero
     */
    public Rational(BigInteger numerator, BigInteger denominator) throws
IllegalArgumentException {
        if (denominator.equals(BigInteger.ZERO)) {
            throw new IllegalArgumentException("Denominator cannot be zero");
        }
        BigInteger gcd = numerator.gcd(denominator);
        this.numerator = numerator.divide(gcd);
        this.denominator = denominator.divide(gcd);
    }

    /**
     * Constructs a Rational number representing an integer value.
     *
     * @param numerator the integer value as a rational number (denominator = 1)
     */
    public Rational(BigInteger numerator) {
        this(numerator, BigInteger.ONE);
    }

    /**
     * Adds this rational number to another rational number and returns the result as
a new Rational object.
     *
     * @param other the Rational number to add
     * @return a new Rational representing the sum of this and the other rational
number
     * @throws NullPointerException if the other Rational is null
     */
```

```java
    public Rational add(Rational other) throws NullPointerException {
        return null;
    }

    /**
     * Subtracts another rational number from this rational number and returns the
result as a new Rational object.
     *
     * @param other the Rational number to subtract
     * @return a new Rational representing the difference between this and the other
rational number
     * @throws NullPointerException if the other Rational is null
     */
    public Rational subtract(Rational other) throws NullPointerException {
        // Subtracts another Rational from this and returns the result
        return null;
    }

    /**
     * Multiplies this rational number by another rational number and returns the
result as a new Rational object.
     *
     * @param other the Rational number to multiply by
     * @return a new Rational representing the product of this and the other rational
number
     * @throws NullPointerException if the other Rational is null
     */
    public Rational multiply(Rational other) throws NullPointerException {
        // Multiplies this Rational by another and returns the result
        return null;
    }

    /**
     * Divides this rational number by another rational number and returns the result
as a new Rational object.
     *
     * @param other the Rational number to divide by
     * @return a new Rational representing the quotient of this and the other rational
number
     * @throws NullPointerException if the other Rational is null
     * @throws ArithmeticException if the other Rational is zero (division by zero)
     */
    public Rational divide(Rational other) throws NullPointerException,
ArithmeticException {
        // Divides this Rational by another and returns the result
        return null;
    }
```

```java
    /**
     * Compares this rational number to another rational number.
     *
     * @param other the Rational number to compare with
     * @return a negative integer if this is less than the other, zero if equal, or a
positive integer if greater
     * @throws NullPointerException if the other Rational is null
     */
    public int compareTo(Rational other) throws NullPointerException {
        // Compares two Rational numbers
        return 0;
    }

    /**
     * Checks if this rational number is equal to another object.
     * Two Rational numbers are considered equal if they have the same numerator and
denominator
     * in their reduced form. Since the constructor automatically reduces the
fraction,
     * equality can be checked directly by comparing the numerator and denominator.
     *
     * @param obj the object to compare with
     * @return true if the other object is a Rational with the same value as this,
false otherwise
     */
    @Override
    public boolean equals(Object obj) {
        // Checks equality between this Rational and another object
        return false;
    }

    /**
     * Returns a hash code for this rational number.
     * The hash code is computed based on both the numerator and denominator in their
reduced form.
     * This ensures that two Rational numbers with the same value (and thus equal by
equals())
     * will have the same hash code.
     *
     * @return an integer hash code for this rational number
     */
    @Override
    public int hashCode() {
        // Provides a hash code for the Rational
        return 0;
    }

    /**
```

```
     * Converts this rational number to a string in the format "numerator/denominator"
or "numerator" if the denominator is 1.
     *
     * @return a string representation of this rational number
     */
    @Override
    public String toString() {
        // Converts the Rational to a string
        return "";
    }

    /**
     * Returns the numerator of this rational number.
     *
     * @return the numerator as a BigInteger
     */
    public BigInteger getNumerator() {
        // Returns the numerator
        return null;
    }

    /**
     * Returns the denominator of this rational number.
     *
     * @return the denominator as a BigInteger
     */
    public BigInteger getDenominator() {
        // Returns the denominator
        return null;
    }

    /**
     * Returns the decimal value of this rational number as a double, with possible
precision loss.
     *
     * @return the decimal approximation of this rational number
     */
    public double toDouble() {
        // Converts the Rational to a double approximation
        return 0.0;
    }

}
```