```java
/*
 * Copyright (C) 2016 redxef.
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
 * MA 02110-1301  USA
 */
package powerspy.baselib;

import java.io.IOException;
import java.io.InputStream;
import java.util.LinkedList;
import java.util.List;

/**
 *
 * @author redxef
 */
public class ArrayInputStream extends InputStream {

        private static final int buffsize = 8;
        private final List<Byte[]> data;
        private int write_pos;
        private int read_pos;

        /**
         * Generates a new ArrayInputStream. The data of this ArrayInputStream
         * can be set via this.insert(). When reading from this InputStream, the
         * data will be read in the same order.
         *
         */
        public ArrayInputStream()
        {
                data = new LinkedList<>();
                data.add(new Byte[buffsize]);
                write_pos = 0;
                read_pos = 0;
        }

        /**
         * Inserts a byte[] of data into the InputStream. b is the data, data
         * will be injected include offset until element offset+leng-1.
         *
         * @param b    the data
         * @param offs the offset from which to begin reading
```

```java
	 * @param leng the number of bytes to read
	 */
	public synchronized void insert(byte b[], int offs, int leng)
	{
		for (int i = offs; i < offs + leng; i++) {
			int pos = write_pos;
			while (pos >= buffsize)
				pos -= buffsize;

			if (data.size() == write_pos / buffsize)
				data.add(new Byte[buffsize]);
			data.get(write_pos / buffsize)[pos] = b[i];
			write_pos++;
		}
	}

	@Override
	public int available()
	{
		return write_pos - read_pos;
	}

	@Override
	public synchronized int read() throws IOException
	{
		int ret = -1;
		if (read_pos < write_pos) {
			int pos = read_pos;
			while (pos >= buffsize)
				pos -= buffsize;
			ret = data.get(read_pos / buffsize)[pos];
			read_pos++;
			if (read_pos == buffsize) {
				data.remove(0);
				read_pos = 0;
				write_pos -= buffsize;
			}
		}

		return ret;
	}

	@Override
	public synchronized long skip(long n)
	{
		long k = write_pos - read_pos;
		if (n < k)
			k = n < 0 ? 0 : n;

		read_pos += k;
		return k;
	}

	@Override
	public boolean markSupported()
	{
		return false;
```

```
        }
}
```