## Converting Octets to Chars

Files and devices use octets. Byte streams are more suitable.

Java uses Unicode chars. Char streams are more suitable.

Need adapter classes that do the conversion.

`InputStreamReader` is a subclass of `Reader` that gets octets from an `InputStream` and converts them to chars.

- `InputStreamReader(InputStream in)`

  Uses the locale encoding.

- `InputStreamReader(InputStream in, String enc) throws UnsupportedEncodingException`

  Uses the named encoding.

E.g.,

```
Reader r = new InputStreamReader(System.in);
r = new BufferedReader(r);
```



## Converting Chars to Octets

`OutputStreamWriter` is a subclass of `Writer` that converts chars to octets and writes them out to an `OutputStream`.

- `OutputStreamWriter(OutputStream out)`

- `OutputStreamWriter(OutputStream out, String enc) throws UnsupportedEncodingException`

E.g.,

```
Writer w = new OutputStreamWriter(System.out);
w = new BufferedWriter(w);
PrintWriter pw = new PrintWriter(w);
```



## String

Strings in Java are objects of class `String`.

`String` objects are *immutable* objects: no method modifies an existing string. Instead, there are methods to create a new string based on an existing one:

- `String(String value)`: copy constructor

- `String concat(String s)` and the + and += operators

- `String substring(int b, int e)`

- `String toLowerCase()`, `String toUpperCase()`, etc.

Some other methods query the string:

- `int length()`

- `char charAt(int index)`, `char[] toCharArray()`

- `boolean equals(Object s)`, `int compareTo(String s)`

There are other methods. Consult the documentation.



## StringBuffer

It may be inefficient to use immutable objects as intermediate values.

`StringBuffer` is *mutable*. You can add characters to it quickly.

When you are done, call `toString()` to get the result.

- `StringBuffer append(...)`

- `StringBuffer insert(...)`

- `StringBuffer reverse()`

- `void setCharAt(int index, char c)`

- `char charAt(int index)`

- `int length()`

There are other methods. Consult the documentation.

## Concurrency

Concurrency in Java is realized by *multi-threading*.

A *thread* is an execution sequence in a running program. A running program can have several threads. They are executed in parallel.

When a program starts, a thread is created to execute `main`. You create other threads by creating objects of type `Thread`:

1. A `Thread` object is created. (No execution yet.)
2. Optionally, its parameters (name, priority, etc.) are set.
3. Its `void start()` method is called.
4. Execution begins at its `void run()` method.
5. Its `run` method terminates. The thread thus terminates.

Thus we want to subclass `Thread` and override `run` to insert our code.



## Thread Example

```
/* From _The Java Programming Language_ */
public class PingPong extends Thread {
  private String word;
  private int delay;
  public PingPong(String w, int d) {
    word = w;
    delay = d;
  }
  public void run() {
    try {
      for (;;) {
        System.out.print(word + " ");
        System.out.flush();
        sleep(delay);
      }
    }
    catch (InterruptedException e) {}
  }
  public static void main(String[] a) {
    new PingPong("ping", 33).start();
    new PingPong("PONG", 100).start();
  }
}
```



## Alternative Thread Creation

Sometimes we do not want our class to be a subclass of `Thread`, e.g., our class has another superclass, or we do not want our class to inherit `Thread` methods.

Then we can use the `Runnable` interface:

1. Our class implements `Runnable` and overrides its `run` method.
2. Create a `Thread` object by passing our object to the constructor.

   `Thread(Runnable r)`

3. Use the `Thread` object as usual. (E.g., we can call its `start` method.)
4. When execution begins, `Thread`'s `run` will call our `run`.



## Alternative Example

```
public class PingPong implements Runnable {
  private String word;
  private int delay;
  public PingPong(String w, int d) {
    word = w;
    delay = d;
  }
  public void run() {
    try {
      for (;;) {
        System.out.print(word + " ");
        System.out.flush();
        Thread.sleep(delay);
      }
    }
    catch (InterruptedException e) {}
  }
  public static void main(String[] a) {
    new Thread(new PingPong("ping", 33))
      .start();
    new Thread(new PingPong("PONG", 100))
      .start();
  }
```