DownloadsTableModel, ProgressRenderer и DownloadManager. Класс Download Manager отвечает за графический интерфейс и предоставляет возможность классам DownloadsTableModel и ProgressRenderer отображать текущий список загрузок. Класс Download обеспечивает управление загрузками и отвечает за процесс загрузки файла. В следующих разделах будут подробно рассмотрены эти классы, описана из работа и взаимодействие.

Класс Download

Класс Download является основным рабочим классом в менеджере загрузок. Его главной задачей является загрузка файла и сохранение его содержимого на диске. Каждый раз, когда новая загрузка добавляется в менеджер загрузок, создается новый объект типа Download для обработки нового файла.

Менеджер загрузок способен загружать несколько файлов одновременно. Для достижения этого необходимо, чтобы каждая новая загрузка производилась независимо от других. Также необходимо иметь возможность управлять состоянием отдельной загрузки таким образом, чтобы ее характеристики отображались в GUI. Все это должен обеспечивать класс Download.

Полностью код для класса Download приведен ниже. Обратите внимание, что класс Download расширяется с помощью класса Observable и реализует интерфейс Runnable. Каждая часть класса подробно исследуется в оставшейся части главы.

```
import java.io.*;
import java.net.*;
import java.util.*;

// Этот класс загружает файл с указанного URL.
class Download extends Observable implements Runnable {
  // Max size of download buffer.
  private static final int MAX_BUFFER_SIZE = 1024;

  // Имена состояния.
  public static final String STATUSES[] = {"Downloading",
    "Paused", "Complete", "Cancelled", "Error"};

  // Коды состояния.
  public static final int DOWNLOADING = 0;
  public static final int PAUSED = 1;
  public static final int COMPLETE = 2;
  public static final int CANCELLED = 3;
  public static final int ERROR = 4;

  private URL url;         // Загрузка URL.
  private int size;        // Размер загрузки в байтах.
  private int downloaded; // Количество загруженных байтов.
  private int status;      // Текущее состояние загрузки.

  // Конструктор для класса Download.
  public Download(URL url) {
    this.url = url;
    size = -1;
    downloaded = 0;
    status = DOWNLOADING;
```

```
    // Начало загрузки.
    download();
  }

  // Получить унифицированный указатель информационного ресурса (URL).
  public String getUrl() {
    return url.toString();
  }

  // Получить размер загрузки.
  public int getSize() {
    return size;
  }

  // Получить процесс загрузки.
  public float getProgress() {
    return ((float) downloaded / size) * 100;
  }

  // Получить состояние загрузки.
  public int getStatus() {
    return status;
  }

  // Приостановить загрузку (пауза).
  public void pause() {
    status = PAUSED;
    stateChanged();
  }

  // Повторить загрузку.
  public void resume() {
    status = DOWNLOADING;
    stateChanged();
    download();
  }

  // Прекратить загрузку.
  public void cancel() {
    status = CANCELLED;
    stateChanged();
  }

  // Отметить загрузку как имеющую ошибку.
  private void error() {
    status = ERROR;
    stateChanged();
  }

  // Начать или повторить загрузки.
  private void download() {
    Thread thread = new Thread(this);
    thread.start();
  }

  // Получить имя файла из URL.
  private String getFileName(URL url) {
    String fileName = url.getFile();
    return fileName.substring(fileName.lastIndexOf('/') + 1);
  }
```

```
// Загрузить файл.
public void run() {
  RandomAccessFile file = null;
  InputStream stream = null;
  try {
    // Открыть соединение с URL.
    HttpURLConnection connection =
      (HttpURLConnection) url.openConnection();

    // Определить часть загруженного файла.
    connection.setRequestProperty("Range",
      "bytes=" + downloaded + "-");

    // Соединиться с сервером.
    connection.connect();

    // Убедиться, что код находится в диапазоне 200.
    if (connection.getResponseCode() / 100 != 2) {
      error();
    }

    // Проверить допустимую длину содержимого.
    int contentLength = connection.getContentLength();
    if (contentLength < 1) {
      error();
    }

    /* Установить размер для загрузки,
       если он еще не установлен. */
    if (size == -1) {
      size = contentLength;
      stateChanged();
    }

    // Открыть файл и найти конец файла.
    file = new RandomAccessFile(getFileName(url), "rw");
    file.seek(downloaded);
    stream = connection.getInputStream();
    while (status == DOWNLOADING) {
      /* Размер буфера в соответствии с размером
         части файла, который осталось загрузить. */
      byte buffer[];
      if (size - downloaded > MAX_BUFFER_SIZE) {
        buffer = new byte[MAX_BUFFER_SIZE];
      } else {
        buffer = new byte[size - downloaded];
      }

      // Считать с сервера в буфер.
      int read = stream.read(buffer);
      if (read == -1)
        break;

      // Записать буфер в файл.
      file.write(buffer, 0, read);
      downloaded += read;
      stateChanged();
    }

    /* Изменить статус завершения, если эта точка уже
       достигнута, поскольку загрузка закончена. */
```

```
      if (status == DOWNLOADING) {
        status = COMPLETE;
        stateChanged();
      }
    } catch (Exception e) {
      error();
    } finally {

      // Закрыть файл.
      if (file != null) {
        try {
          file.close();
        } catch (Exception e) {}
      }

      // Закрыть соединение с сервером.
      if (stream != null) {
        try {
          stream.close();
        } catch (Exception e) {}
      }
    }
  }

  // Уведомить, что статус этой загрузки был изменен.
  private void stateChanged() {
    setChanged();
    notifyObservers();
  }
}
```

Переменные загрузки

Загрузка начинается в объявления нескольких статических переменных, которые определяют различные состояния, используемые классом. Затем объявляются четыре переменные. Переменная `url` содержит URL для файла, который должен быть загружен. Переменная `size` содержит размер загружаемого файла в байтах. Переменная `downloaded` содержит количество байтов, которые уже загружены к этому времени, а переменная `status` содержит текущее состояние загрузки.

Конструктор Download

В конструктор менеджера загрузки передается ссылка на URL, с которого будет происходить загрузка, в форме объекта URL, который связан с переменной `url`. Затем в конструкторе инициализируются переменные и вызывается метод `download()`. Обратите внимание, для переменной `size` устанавливается значение `-1`, свидетельствующее о том, что размер еще не установлен.

Метод download()

В методе `download()` создается новый объект типа `Thread` и ему передается ссылка на вызывающий экземпляр `Download`. Как уже упоминалось, это необходимо сделать для того, чтобы каждая загрузка выполнялась независимо от других загрузок. Для того чтобы класс `Download` работал независимо, он должен выполняться

Метод showPage()

Метод showPage() загружает и отображает страницы в панели редактора мини-Web-браузера. Поскольку в этом методе обрабатывается много действий, производится последовательная проверка строки за строкой. Метод начинается следующими строками кода.

```
private void showPage(URL pageUrl, boolean addToList) {
  // Показать песочные часы на время работы червя.

  setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
```

В этот метод передается URL страницы, которую необходимо отобразить, и флаг, на основании которого делается вывод о добавлении страницы в список страниц. Затем для курсора устанавливается значение WAIT_CURSOR, говорящее о том, что приложение занято. В большинстве операционных систем значение WAIT_CURSOR соответствует отображению песочных часов. После установки курсора указанная страница отображается в панели редактора, как показано в листинге.

```
try {
  // Получить URL страницы для отображения.
  URL currentUrl = displayEditorPane.getPage();
  // Загрузить и отобразить заданную страницу.
  displayEditorPane.setPage(pageUrl);
  // Получить URL новой страницы для отображения.
  URL newUrl = displayEditorPane.getPage();
```

Перед загрузкой новой страницы в панель редактора, URL страницы, которая в это время отображается, сохраняется. Текущая страница записывается, чтобы она могла быть использована позже в этом методе, если она еще не добавлена в список страниц. Затем загружается новая страница. После загрузки новой страницы, ее URL также заносится в список страниц для использования в методе.

В следующих нескольких строках кода производится проверка на предмет того, что страница, передаваемая в метод showPage(), должна быть добавлена в список страниц для последующего использования.

```
// Если список определен, то добавить страницу.
if (addToList) {
  int listSize = pageList.size();
  if (listSize > 0) {
    int pageIndex = pageList.indexOf(currentUrl.toString());
  if (pageIndex < listSize -1) {
    for (int i = listSize -1; i > pageIndex; i--) {
      pageList.remove(i);
    }
  }
}
pageList.add(newUrl.toString());
}
```

Если установлен флаг addToList, то отображаемая страница добавляется в список страниц. Сначала извлекается размер списка страниц. Если в списке есть хоть одна страница, извлекается индекс последней отображаемой страницы. Если индекс меньше чем размер списка, тогда все страницы в списке, расположенные после последней отображаемой страницы, удаляются. Это происходит потому, что впереди не должно быть страниц для отображения.

Затем обновляется графический интерфейс пользователя, как показано ниже.

```
// Записать в текстовое поле URL текущей страницы.
locationTextField.setText(newUrl.toString());
// Обновить кнопки в соответствии с отображаемой страницей.
updateButtons();
```

Сначала обновляется текстовое поле для отображения URL страницы, которая будет отображаться в браузере. Затем вызывается метод `updateButtons()` для получения активного или пассивного состояния кнопок Back и Forward в зависимости от положения записи о странице в списке страниц.

Если при загрузке новой страницы возникнет исключительная ситуация, выполняется блок `catch`, как показано ниже.

```
}
catch (Exception e)
{
  // Отобразить сообщение об ошибке.
  showError("Unable to load page"); ;
}
```

Если возникает исключительная ситуация, то вызывается метод `showError()` для отображения диалогового окна с сообщением об ошибке.

В методе `showPage()` используются расширенные возможности блока `try...catch` с ключевым словом `finally`. Конструкция `finally` используется для того, чтобы в любом случае возвратить курсор в его состояние по умолчанию.

```
finally
{
  // Возвратить курсор по умолчанию.
  setCursor(Cursor.getDefaultCursor());
}
```

Метод updateButtons()

Метод `updateButtons()`, листинг которого приведен ниже, обновляет состояние кнопок Back и Forward в панели кнопок на основе положения записи о текущей отображаемой странице в списке страниц. Этот метод вызывается в методе `showPage()` каждый раз, когда производится отображение страницы.

```
/* Обновить состояние кнопок Back и Forward в
   соответствии с отображаемой страницей. */
private void updateButtons() {
  if (pageList.size() < 2) {
    backButton.setEnabled(false);
    forwardButton.setEnabled(false);
  }
  else {
    URL currentUrl = displayEditorPane.getPage();
    int pageIndex = pageList.indexOf(currentUrl.toString());
    backButton.setEnabled(pageIndex > 0);
    forwardButton.setEnabled(
      pageIndex < (pageList.size() - 1));
  }
}
```

Если в списке меньше, чем две страницы, обе кнопки Back и Forward будут пассивными и будут отображаться в приглушенном сером цвете. Это происходит пото-