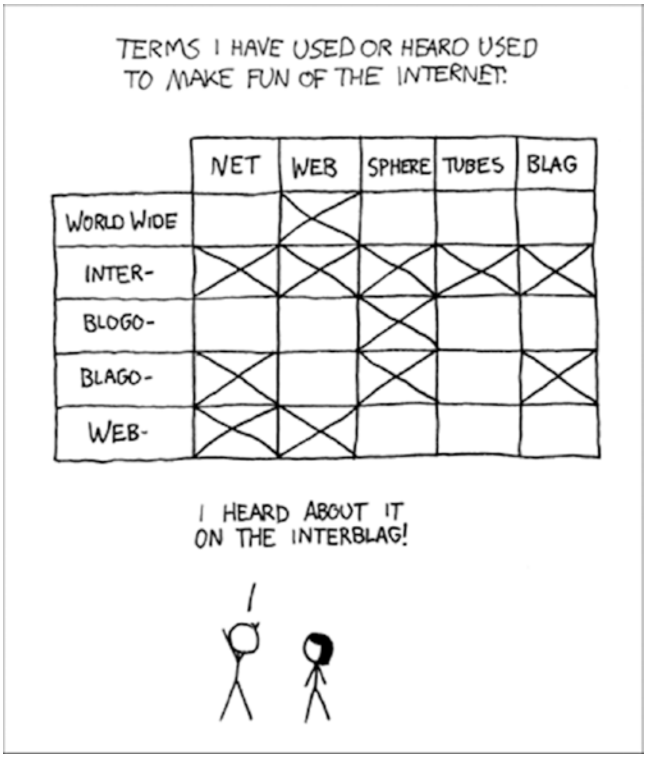
CHAPTER 17

The Interweb


TERMS I HAVE USED OR HEARD USED TO MAKE FUN OF THE INTERNET:
NET
WEB
SPHERE
TUBES
BLAG
WORLD WIDE
INTER-
BLOGO-
BLAGO-
WEB-
I HEARD ABOUT IT ON THE INTERBLAG!

(Courtesy xkcd: Interblag)

17.1 Web 101

The Web is a complex beast. Here's what you need to know:

- *Server*: The computer serving web pages
- *Client*: The computer that receives web pages and is used by a person
- *HTML*: The language used to create web pages
- *CSS*: "Cascading style-sheets"; store the styles of the web page
- *JavaScript*: A programming language that is used within web pages and executed on the client

17.2 My First Web App

You should make something very basic for your first web application. This way, you will have a better understanding of what's going on "behind the scenes" of many web frameworks.

Create a file called `App.java` and copy the following code into it:

```java
import java.io.IOException;
import java.io.OutputStream;
import java.net.InetSocketAddress;
import com.sun.net.httpserver.*;

public class App {

    static class MyHandler implements HttpHandler {
        public void handle(HttpExchange t) throws IOException {
            String response = "<html> Hello Inter-webs! </html>";
            t.sendResponseHeaders(200, response.length());
            OutputStream os = t.getResponseBody();
            os.write(response.getBytes());
            os.close();
        }
    }

    public static void main(String[] args) throws Exception {
        HttpServer server = HttpServer.create(new InetSocketAddress(8000), 0);
        server.createContext("/", new MyHandler());
        server.setExecutor(null); // creates a default executor
        server.start();
        System.out.println("Server running at http://localhost:8000/");
    }

}
```

All this does is create an `HttpServer` that listens for connections on port 8000 and responds with a message.

After running this code (`javac App.java && java App`), open your web browser and point it to `http://localhost:8000/`.

`localhost` refers to the computer you're on, and `:8000` refers to port 8000.

Congratulations! You just made a web application!

It's not on the Internet yet, and it's extremely simple, but it's a good start.

PORT?

- *URL (Universal Resource Locator)*: The unique name used to locate resources on any network or machine. Sometimes it starts with "http"; sometimes it includes a port.
- *HTTP Hypertext Transfer Protocol*: The typical protocol used to communicate over the wire
- *Port*: A number that must be specified when communicating between computers (the default port for HTTP is 80)

17.3 The Holy Grails

Grails is a web framework for Groovy that follows the example of Ruby on Rails (hence *Grails*). It is an opinionated web framework with a command-line tool that gets things done really fast. Grails uses convention over configuration to reduce configuration overhead.

Grails lives firmly in the Java ecosystem and is built on technologies such as Spring and Hibernate. Grails also includes an object-relational mapping (ORM) framework called *GORM* and has a large collection of plug-ins.

17.3.1 Quick Overview

After installing Grails,[1] you can create an app by running the following on the command line:

```
$ grails create-app
```

[1]This overview is based on Grails 2.1.4, but the basics should remain the same for all versions of Grails.

Then, you can run commands such as `create-domain-class` and `generate-all` to create your application as you go. Run `grails help` to see the full list of commands available.

Grails applications have a very specific project structure. The following is a simple breakdown of *most* of that structure:

- `grails-app`**:** The Grails-specific folder
    - conf: Configuration, such as the data source and Bootstrap
    - controllers: Controllers with methods for index/create/edit/delete, or anything else
    - domain: Domain model; classes representing your persistent data
    - i18n: Message bundles
    - jobs: Any scheduled jobs you might have go here
    - services: Back-end services in which your back end or "business" logic goes
    - taglib: You can very easily define your own tags for use in your GSP files.
    - views: Views of MVC; typically, these are GSP files (HTML-based)
- `src`**:** Any utilities or common code that doesn't fit anywhere else
    - java: Java code
    - groovy: Groovy code
- `web-app`
    - css: CSS style sheets
    - images: Images used by your web application
    - js: Your JavaScript files
    - WEB-INF: Spring's `applicationContext.xml` goes here.

To create a new domain (model) class, run the following:

```
$ grails create-domain-class
```

It's a good idea to include a package for your domain classes (such as `example.Post`).

A domain class in Grails also defines its mapping to the database. For example, here's a domain class representing a blogpost (assuming User and Comment have already been created):

```
class Post {
    String text
   int rating
    Date created = new Date()
    User createdBy

    static hasMany = [comments: Comment]

    static constraints = {
        text(size:10..500)
    }
}
```

The static `hasMany` field is a map that represents one-to-many relationships in your database. Grails uses `Hibernate` in the background to create tables for all your domain classes and relationships. Every table gets an `id` field for the primary key by default.

To have Grails automatically create your controller and views, run the following:

```
$ grails generate-all
```

Grails will ask if you want to overwrite existing files, if they exist. So, be careful when using this command.

When you want to test your app, you simply run the following:

```
$ grails run-app
```

When you're ready to deploy to a server, you can create a "war" file by typing this:

```
$ grails war
```

17.3.2 Plug-ins

The Grails ecosystem now includes over 1,000 plug-ins. To list all of the plug-ins, simply execute

```
$ grails list-plugins
```

When you've picked out a plug-in you want to use, execute the following (with the plug-in name and version):

```
$ grails install-plugin [NAME] [VERSION]
```

This will add the plug-in to your project. If you decide to uninstall it, simply use the `uninstall-plugin` command.

Only an Overview This has been only a brief overview of Grails. Many books have been written about Grails and how to use it. For more information on using Grails, please visit `grails.org`.[2]

17.4 Cloud

Grails is supported by the following cloud providers:

- CloudFoundry[3]
- Amazon[4]
- Heroku[5]

However, it is not within the scope of this book to go over all of them, so I'll talk talk about *Heroku*.

Heroku[6] is owned by Salesforce.com.[7] It was one of the first cloud platforms and has been in development since June 2007. When it began, it supported only Ruby, but it has since added support for Java, Scala, Groovy, Node.js, Clojure, and Python. Heroku supports multiple tiered accounts, including a free account.

Heroku relies on `git` for pushing changes to your server. For example, to create an app in Heroku using the CLI, do the following:

```
$ heroku create
$ git push heroku master
```

Your app will be up and running, and Heroku will identify the URL where you will find it.

Go launch a Grails app on Heroku!

[2]`http://grails.org/`.
[3]`www.cloudfoundry.com/`.
[4]`http://aws.amazon.com/ec2/`.
[5]`www.heroku.com/`.
[6]`www.heroku.com/`.
[7]`http://news.heroku.com/news_releases/salesforcecom-signs-definitive-agreement-to-acquire-heroku`.

17.5 The REST

REST stands for REpresentational State Transfer.[8] It was designed in a Ph.D. dissertation and has gained some popularity as the new web-service standard. Many developers have praised it as a much better standard than SOAP (which I will not attempt to describe).

At the most basic level in REST, each *CRUD* (create, read, update, delete) operation is mapped to an HTTP method.

- Create: `POST`
- Read: `GET`
- Update: `PUT`
- Delete: `DELETE`

The transport mechanism is assumed to be HTTP, but the message contents can be of any type, usually XML or JSON.

The JSR community has designed the JAX-RS API for building RESTful Java web services, while Groovy and Scala both have some built-in support for XML and JSON and various way of building web services.

17.5.1 Using Maven Archetypes

You can create a simple Java REST (JAX-RS) application using Maven, as follows:

```
mvn archetype:generate
```

Wait for things to download and then choose "tomcat-maven-archetype" (type "tomcat-maven" and press Enter, then type "1"; Enter; Enter). You will need to enter a groupId and artifactId.

After creating your application, you can start it by typing the following command:

```
mvn tomcat:run
```

17.6 Summary

Congratulations! You now understand the Interweb. Yes, it *is* a series of tubes. Ted Stevens (see following) was right!

> *...They want to deliver vast amounts of information over the Internet. And again, the Internet is not something that you just dump something on. It's not a big truck. It's a series of tubes. And if you don't understand, those tubes can be filled and if they are filled, when you put your message in, it gets in line and it's going to be delayed by anyone that puts into that tube enormous amounts of material, enormous amounts of material.*
>
> —Theodore "Ted" Fulton Stevens, Sr.—US senator from Alaska from December 24, 1968–January 3, 2009

[8]`www.ics.uci.edu/~fielding/pubs/dissertation/top.htm`.

CHAPTER 18

Swinging Graphics

Swing is the Java API for building cross-platform GUIs (graphical user interfaces).

If you ever want to write a graphical program (a computer game, for example), you will have to use Swing or some similar API.

There are many other libraries for doing graphics in Java, but Swing is built in.

All of the code for this chapter can be found on the GitHub repository.[1]

18.1 Hello Window

The most basic concept of graphics is getting stuff onto the screen.

The easiest way to do this in Swing is to use `JWindow`, for example:

```
import javax.swing.*;

public class HelloWindow extends JWindow {

        public HelloWindow() {
                setSize(500, 500); //width, height
                setAlwaysOnTop(true);
                setVisible(true);
        }

        @Override
        public void paint(Graphics g) {
                g.setFont(g.getFont().deriveFont(20f));
                g.drawString("Hello Window", 10, 20); //x,y
        }

```

[1] https://github.com/modernprog/part3.

A. L. Davis, *Modern Programming Made Easy*, DOI 10.1007/978-1-4842-2490-8_18

```
        public static void main(String[] args) {
                new HelloWindow();
        }

}
```

Running this code will create a window at the top left of your screen, with the words "Hello Window" printed on it.

In the constructor, the following occurs:

- The width and height of the window are both set to 500 pixels.
- The window is set to always be displayed (above all other windows) with the `setAlwaysOnTop` method.
- Finally, `setVisible(true)` is called to make the window visible.

The `paint` method gets called every time the window is drawn on the screen. This method simply does the following:

- Sets the font size to 20
- Draws the string "`Hello World`" at coordinates x=10, y=20 (coordinates are always in pixels)

You might notice that the "window" doesn't have any edges, header, menu, or minimize/maximize icons that you're used to. To get these things, you use a `JFrame`. Here's a very simple example:

```
import javax.swing.*;

public class HelloFrame extends JFrame {

        public HelloFrame() {
                super("Hello");
                setSize(500, 500);
                setAlwaysOnTop(true);
                setVisible(true);
                setDefaultCloseOperation(EXIT_ON_CLOSE);
        }

        public static void main(String[] args) {
                new HelloFrame();
        }

}
```

Running this code creates a 500×500 "window with frame" with the name "Hello," and closing the window would exit the application.

18.2 Push My Buttons

Buttons are one of the ways that users can interact with your program. To cause something to happen when a button is pressed, you use an "ActionListener," for example:

```
JButton button = new  JButton("Talk!");
button.addActionListener(new   ActionListener() {
       public void actionPerformed(ActionEvent e) {
               JOptionPane.showMessageDialog(HelloFrame.this, "Hello!");
        }
});
getContentPane().add(button);
```

The `showMessageDialog` method of `JOptionPane` is similar to JavaScript's `alert` method, in that it shows a pop-up window.

In Java 8, the `ActionListener` part can be shortened to the following:

```
button.addActionListener(e -> JOptionPane.showMessageDialog(this, "Hello!"));
```

The Groovy syntax is slightly different (it only requires a `{ and }`).

```
button.addActionListener({e ->JOptionPane.showMessageDialog(this, "Hello!")})
```

Swing has many interfaces that end with the word "Listener," such as:

- `KeyListener`
- `MouseListener`
- `WindowListener`

The *Listener* pattern is very similar to the *Observer* design pattern.

18.3 Fake Browser

Let's make a web browser!

Let's begin by creating the fields and constructor for the class, as follows:

```
public class Browser extends JFrame {

        JTextField urlField = new JTextField();
        JEditorPane viewer = new JEditorPane();
        JScrollPane pane = new JScrollPane();

        public Browser() {
                super("Browser");
                setSize(800,600);
                setAlwaysOnTop(true);
                setDefaultCloseOperation(EXIT_ON_CLOSE);
                init();
        }
```

JTextField will be used to input the URL. JEditorPane is used to show the HTML, and the JScrollPane allows the page to be scrollable.

Next, we define the init() method to put everything together.

```
private void init() {
        viewer.setContentType("text/html");
        pane.setViewportView(viewer);
        JPanel panel = new JPanel();
        panel.setLayout(new BorderLayout(2,2));
        panel.add(pane, BorderLayout.CENTER);
        panel.add(urlField, BorderLayout.NORTH);
        getContentPane().add(panel);
        urlField.addKeyListener(new KeyAdapter() {
                @Override
                public void keyReleased(KeyEvent e) {
                        handleKeyPress(e);
                }
        });
}
```

The viewer is set as the viewport view of the JScrollPane, so it can be scrolled.

JPanel is created with a BorderLayout. This allows us to arrange urlField on top of the scroll pane, much as in a real browser. KeyListener is used to call handleKeyPress whenever a key is pressed inside urlField.

Next, we fill out the handleKeyPress method.

```
private void handleKeyPress(KeyEvent e) {
        if (e.getKeyCode() == KeyEvent.VK_ENTER) {
                try {
                        viewer.setPage(new URL(urlField.getText()));
                } catch (MalformedURLException ex) {
                        ex.printStackTrace();
                } catch (IOException ex) {
                        ex.printStackTrace();
                }
        }
}
```

This method simply sets the page of `JEditorPane` to the URL from `urlField` whenever the Enter key is pressed.

Finally, we define the main method.

```
public static void main(String[] args) {
 new  Browser().setVisible(true);
}
```

Eat your own dog food Run your app from Chapter 18. Open your fake browser, and point it to the app at `http://localhost:8000/`.

18.4 Griffon

Griffon[2] is a Groovy-based framework for creating Swing GUIs. It has a command-line interface very similar to Grails.

To begin a new project type, use the following:

```
griffon create-app griffon-example -archetype=jumpstart
```

Here, the name of the project is "griffon-example."

Griffon uses the MVC design pattern and Groovy DSL to make it much easier to build Swing applications.

18.5 Advanced Graphics

Although far beyond the scope of this book, there are several libraries that can be used for 2D or 3D graphics. Here are some of them.

Java 2D

- JavaFX[3]
- JFreeChart[4]
- Piccolo2D[5]
- JMagick[6]

[2]`http://griffon.codehaus.org/`
[3]`www.oracle.com/technetwork/java/javafx/index.html`
[4]`www.jfree.org/jfreechart/`
[5]`www.piccolo2d.org/`
[6]`http://sourceforge.net/projects/jmagick/`

Java 3D

- JOGL[7]
- JMonkeyEngine[8]

JavaScript 2D

- d3[9]
- Highcharts[10]

JavaScript 3D

- three.js[11]

18.6 Graphics Glossary

Component: Any graphical element defined in the Java graphics API

Double-Buffering: A technique used in graphics in which elements are drawn in memory before being sent to the computer screen. This avoids flicker.

Frame: In Swing, the frame (JFrame) is used to represent what we typically call a "window" in the GUI.

GUI: Graphical user interface

Layout: Strategy used by Swing for arranging components within a panel or other component

Menu: There are two kinds of menus: a windows built-in menu (JMenu) and a pop-up menu (JPopupMenu).

Menu item: In Swing, the `JMenuItem` represents one line of a menu that can have an action associated with it.

Panel: In Swing, `JPanel` is used to contain other components.

Pixel: Smallest unit of the screen that is drawable. A typical screen has millions of pixels that are arranged in a grid.

Window: Rectangular section of the screen. In Swing, the Window object has no border, so it can be used for a splash image, for example.

[7]`http://download.java.net/media/jogl/www/`.
[8]`http://jmonkeyengine.org/`.
[9]`http://d3js.org/`.
[10]`www.highcharts.com/`.
[11]`http://threejs.org/`.

18.7 Summary

You just learned the following:

- Creating a cross-platform GUI interface in Java and Groovy
- How to make a web browser worse than IE
- Some of the available graphics libraries

CHAPTER 19

Creating a Magical User Experience

First, you should be aware of the following acronyms:

UX: User experience

UI: User interface

KISS: Keep it simple, stupid

19.1 Application Hierarchy

You should prioritize your UX according to the following characteristics, from highest to lowest:

1. Functionality: Software does what it should.
2. Usefulness: Is the software easy to use?
3. Efficiency: Can the user work efficiently?
4. Magicalness: Is the experience magical?

You can't focus on being usable if your software is not functional. You can't focus on being efficient if your software is not usable.

After you have mastered all the basics (functionality, usability, and efficiency), only then can you attempt to make your UI magical.

19.2 Consider Your Audience

It's always important to consider the audience for your software. You should get to know them.

A. L. Davis, *Modern Programming Made Easy*, DOI 10.1007/978-1-4842-2490-8_19

Those of you familiar with Harry Potter (or magic, in general) will recognize the words *wizard/witch* and *muggle*. In Potter's world, a *Squib* is someone who's aware of magic but not able to practice it, and a muggle is a normal person who's unware of magic.

We can apply this analogy to software.

- *Random User*: Muggle
- *Rock Star*: Squib
- *Genius*: Wizard/witch

19.3 Choice Is an Illusion

The more choices a person has, the more thinking they are required to do. As a designer, you should do the following:

- Limit choices
- Prepare for every possible choice
- Tailor choices for your audience

You will often have to decide whether to give your user a choice or make the choice for them.

The easy way is always to let the user decide, but the better way is generally to give the user one less choice. This will make your software simpler and, therefore, easier to use.

19.4 Direction

Work instinctively—instinct is your friend. Motion is a subtle and effective way of getting the user's attention. Of course, too much motion is a distraction, so it should be kept to a minimum.

Another instinctual image is the human face. Faces are noticed first. This is why you always see faces on the left-hand side of text (in languages that read from left to right). The eye is drawn first to the face and then to the accompanying text.

19.5 Skuemorphism

Skuemorph: is something from real life that is imitated in software.

Simulating real-life features, such as edges, bevels, and buttons, can be useful for communicating *affordability* (what the user can do with something). However, you have to get it 100% right, if you're simulating a complete object (such as a book). This is why skuemorphism is generally a bad idea. Imitating something from the real world comes across as fake, if it is not done perfectly.

Windows 8 exemplifies the opposite approach—it attempts to remove all metaphor. The new UI of Windows (sometimes called Metro) is very flat and edgeless. Of course, you can take this concept too far. For example, what is clickable should still be obvious.

19.6 Context Is Important

Three stars with no context could mean anything. However, given context (3/5 stars), what is meant becomes obvious.

Also, context is important for navigation. It must be obvious where the user is in your software at all times and how to navigate somewhere else. Otherwise, your users will feel lost, which is not a comfortable feeling.

A related concept is to avoid "modes." The more ways there are to interact with the software, the more complex it will seem.

19.7 KISS

Above all, keep things simple—simple for the user. For example, in general, there should always be one way to do something in the software. Also, as a general rule, your UI should follow the conventions set by existing software/web sites (for example, always underline links).

As your software grows, you will constantly have to make choices about new UI features. In addition to other considerations, you should also contemplate how they can be made simpler.

19.8 You Are Not the User

Unless you are building software only for yourself, the overwhelming probability is that your users are very different from you. For this reason, you must not only try to think like your user but also really get to know him or her.

This means ideally that you sit down and watch your users operate the software.

19.9 Summary

From this chapter, you should have learned the following:

- Your UI should be functional, usable, and efficient, in that order.
- Consider who your user is during all phases of design
- Limit choices and handle all conditions
- Instinct is your friend, but don't imitate reality.
- Keep things simple for users and listen to them

For more about usability, I highly recommend Steve Krug's *Don't Make Me Think* (New Riders, 2014).

CHAPTER 20

Databases

Databases are an extremely important component of most software projects.

If you're not familiar with a database, it is a software system that stores data and enables calculations on those data, somewhat like a spreadsheet.

An original database is known as a *relational databas*e, because it stores relationships between tables in the database. A database typically consists of several highly structured data tables with defined constraints. For example, each column of a table has a type, whether or not it can be null, if it must be unique, and other constraints.

There is a highly standardized language for performing operations and calculations on a database called *SQL* (Structured Query Language). SQL has been around a long time and could easily warrant its own book, so I will only cover the basics here.

Since the advent of so-called big-data projects (such as a particular "face"-themed social network), a second category of databases has emerged: NoSQL databases. Typically, these are more like key-value pairs than relational databases. They include Redis, MongoDB, Cassandra, and many others.

Note The SQL/NoSQL categorization is an oversimplification, but it provides an easier narrative than the actual complex reality. In other words, "Here be dragons!"

20.1 SQL (Relational) Databases

Part of classic relational databases is the concept of ACID[1] (atomicity, consistency, isolation, durability). To summarize ACID, it means that the database is always in a consistent state (with enforced constraints), even if the system crashes in the middle of an update. For example, if a column in a table is marked as "non null," it will never be null. This may seem like a simple thing to achieve, but it is actually very hard.

Some good open source databases include PostgreSQL, MySQL, and H2.

[1]https://en.wikipedia.org/wiki/ACID.

20.1.1 SQL

The basic language of relational databases is SQL. It includes the ability to define tables and perform complex queries on those tables.

For example, creating a table looks something like the following:

```
CREATE TABLE dragon(
    dragon_id INTEGER,
    dragon_name VARCHAR(100),
    birth_date DATE NOT NULL,
    PRIMARY KEY (dragon_id)
   );
```

A table always needs to have a primary key—it acts as the identifier for each row of the table. In this case, the primary key is `dragon_id`.

Database types cover the basics, such as `INTEGER`, but other unfamiliar types include the following:

- `VARCHAR`(length) is similar to the `String` object. It has a given maximum length.
- `TIMESTAMP` is used to store dates and times.
- `NUMERIC(precision, scale)` or `DECIMAL(precision, scale)` is used to store numbers such as currency values (for example, the number 123.45 has a precision of 5 and a scale of 2).
- `BLOB` is typically used to store binary data.

To find the birthday of your oldest dragon, you might perform the following query:

```
SELECT MIN(birth_date) FROM dragon;
```

Or, to select all dragons whose names start with *S* (in alphabetic order), run the following:

```
SELECT dragon_id, dragon_name FROM dragon
    WHERE dragon_name LIKE 'S%'
    ORDER BY dragon_name;
```

20.1.2 Foreign Keys

A *foreign key* is simply a column in a table that references the primary key of another table.

For example, let's say you have a wizard table, and each wizard has multiple dragons they keep as pets. If the wizard table's primary key is `wizard_id`, the "dragon" table would have the following column and foreign key constraint:

```
owner INTEGER,
FOREIGN KEY owner REFERENCES wizard (wizard_id)
```

Although SQL keywords are shown in uppercase, this is not required by all databases.

20.1.3 Connections

A database system typically runs as a separate process, and your code connects to it in some way.

There are many different ways to do this.

In Java, the most basic standard for connecting to databases is called JDBC.

It allows you to run SQL statements on the database. You will need a specific *driver* for your database.

There are also *object-relational mapping* (ORM) frameworks, such as Hibernate.[2] These frameworks have you map Java objects to data tables. They are built *on top of* JDBC. For example, Hibernate has its own query language, called HQL, which is translated into SQL by Hibernate. Grails uses Hibernate.

Alternatively, there are code-generating frameworks that allow you to use a DSL for queries. One such framework for Java is jOOQ.[3] It allows you to write type-safe queries in the native language. For example:

```
create.selectFrom(DRAGON)
  .where(DRAGON.NAME.like("S%"))
  .orderBy(DRAGON.NAME)
```

20.2 NoSQL Databases

Big web projects (such as Wikipedia) had problems using relational databases to scale up to millions of users. They had to partition their database onto multiple machines (called *sharding*), which broke foreign key references. So, over time, big-data projects moved to NoSQL databases, which are basically key-value stores that can be scaled up more easily.

NoSQL databases are used by Netflix, Reddit, Twitter, GitHub, Pinterest, eBay, eHarmony, craigslist, and many others.

■ **Note** I will cover some NoSQL databases here, but there are many others.

[2] http://hibernate.org/orm/.
[3] http://jooq.org/.

20.2.1 Redis

Redis[4] is a key-value store. Everything is stored as a string in Redis, including binary data. It's written in C and has a long list of commands.[5]

There are multiple clients for using Redis from many different languages, including Java, Node.js, Scala, Ruby, Python, and Go.

20.2.2 MongoDB

MongoDB[6] calls itself a document database. It stores JSON-style (JavaScript) documents and has a rich query syntax. It's written in C++, but JavaScript can be used in queries and aggregation functions.

MongoDB supports indexing of any field in a document. It scales horizontally using sharding and provides high availability and increased throughput using replication.

MongoDB can also be used as a file system.

20.2.3 Cassandra

Cassandra[7] was originally developed at Facebook and was released as an open source project in July 2008. It's written in Java and is now a mature, top-level Apache project.

Cassandra is scalable, decentralized, fault-tolerant, and has tunable consistency. It also uses replication for fault-tolerance and performance.

Cassandra has an SQL-like alternative called *CQL* (Cassandra Query Language). Language drivers are available for Java (JDBC), Python (DBAPI2), and Node.JS (Helenus).

20.2.4 VoltDB

VoltDB[8] provides a counter-example to the SQL/NoSQL divide. It's distributed, in-memory, and lightning-fast, but it's also a relational database and supports SQL.

20.3 Summary

- There are two major types of databases: SQL and NoSQL.
- Relational (SQL) databases are highly structured, consistent, and durable, but difficult to scale up.
- Big-data projects tend to use NoSQL databases, which are key-value stores that can be scaled up more easily.

[4]http://redis.io/.
[5]http://redis.io/commands.
[6]www.mongodb.org/.
[7]http://cassandra.apache.org/.
[8]http://voltdb.com/.