Chapter 36. Finding Marvel Comic Covers

Back in February, Marvel released a web API for accessing information about their comics (http://developer.marvel.com/). It's a free service, although you do need to sign up for an account, and use assigned public and private API keys in your programs. The API is quite extensive, with excellent documentation, including an online interactive query test page at http://developer.marvel.com/docs. They also produced a short video about the API which was presented at SXSW 2014 (http://www.youtube.com/watch?v=EKwdUj5h7CY).

Marvel says it has already uploaded information on over 30,000 comics and 7,000 series, with more to follow. For now, at least, Marvel's API can only be used for non-commercial applications, and each user can make at most 3,000 API calls per day.

This article describes a small collection of Java applications I developed for finding comic covers. For example, they allow a user to find the comic cover for the first appearance of the Juggernaut, or the famous issue where Wolverine battled the Hulk. Tracking down those covers inside Marvel's giant database requires a bit of detective work, but the resulting high-quality images (shown in Figure 1) are worth it.

Figure 1. Covers for the First Appearances of the Juggernaut and Wolverine.

I'll start by giving an overview of the Marvel database, then explain how to use my tools to find the two covers in Figure 1, and finish by looking at some of the implementation details of my programs.

If you're more interested in accessing the API from inside a browser, I recommend looking at the articles by Raymond Camden on using Javascript (http://www.raymondcamden.com/index.cfm/2014/2/2/Examples-of-the-Marvel-API and http://java.dzone.com/articles/more-examples-marvel-api). There's also a Node.js wrapper available at https://github.com/fiveisprime/marvel-api, and a blog post about utilizing AngularJS at http://ryanchristiani.com/beginning-angularjs-and-the-marvel-api/.

The Marvel Database

One way of viewing the Marvel API is as a window onto a database of six tables, as depicted in Figure 2.

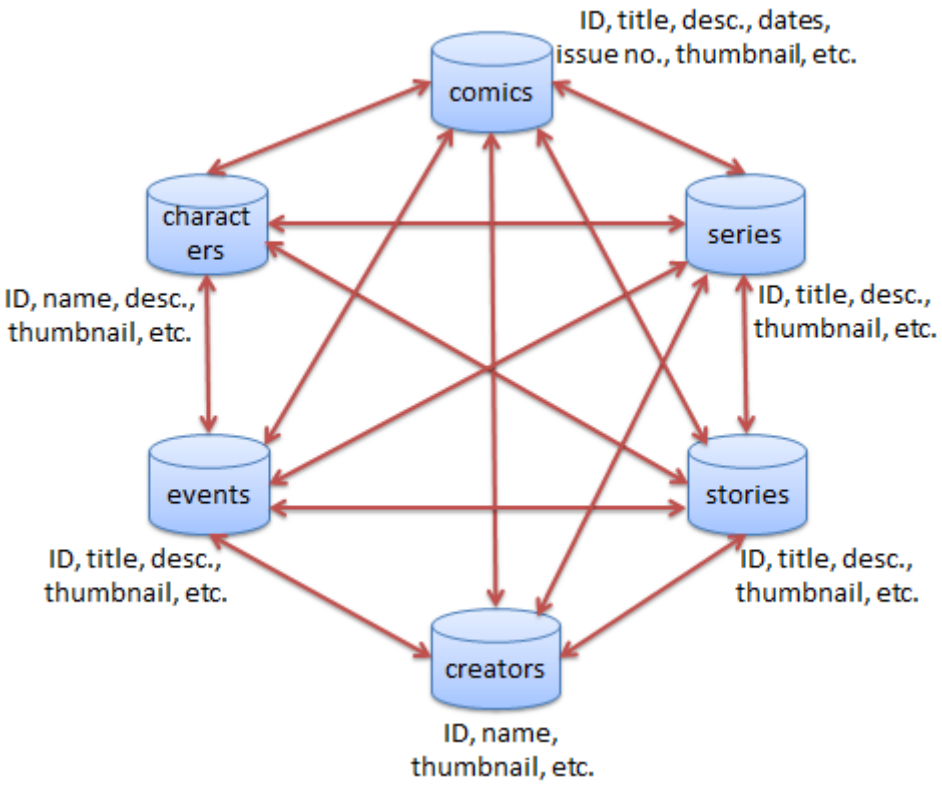


Figure 2. The Six Tables in the Marvel Database.

The six tables are:

- *Comics*: a table holding information on every comic issue, including digital versions, collections and graphic novels. For example: the entry for *Amazing Fantasy* #15 gives its title, on-sale date, page count, and thumbnail image. By the way, although the API calls them "thumbnails", all the images I've retrieved have been quite large—covers seem to be 550 x 830 pixels.
- *Characters*: a table containing data on the people, groups, aliens, animals, and others who inhabit the Marvel Universe. For instance, the details about "Spider-Man", including a brief description and a picture.
- *Series*: information about every comic series, such as *Uncanny X-Men*.
- *Stories*: details about the every Marvel story. Stories are often reprinted, and so a single story may appear in multiple comics.
- Comic *events*: details on the big storylines that span multiple comics and series. For example, "Civil War"
- *Creators*: the people and groups who create the comics, such as Jack Kirby and Stan Lee. My fear is that the thumbnail for *Stan the Man* is http://cdn.bleedingcool.net/wp-content/uploads/2014/03/Stanleenude8242012.jpeg?9098e0. But I was too afraid to check.

Figure 2 shows some of the information available in each table; complete details can be found in the "Entity Types and Representations" documentation at http://developer.marvel.com/documentation/entity_types.

The double-headed arrows between the tables indicates that a query sent to one table can return information from the other tables. For instance, information of a particular comic will include details about the stories in that issue, the characters who appear, the overall series, the comic's creators, and its part in a wider storyline event.

There's an interesting missing connection in Figure 2 – there's no direct link between creators and characters. This means that it isn't possible to look up a given character (e.g. "Spider-man") and find their 'creators'. The nearest you can get is to find the creators involved with the first comic that used the character.

A crucial aspect of searching the database is its use of ID numbers to uniquely identify every entity. This makes sense since character names and series names are rarely unique (e.g. does *The Avengers* mean the original series, one of the later volumes, or the ultimates?). A related problem is deciding on the name of a character. For instance, the database doesn't return any matches when you search for "Aunt May" or "Dr. Doom", only when you look for "May Parker" or "Doctor Doom".

The best way of finding information is to use entity IDs, but how do you find out what they are? Yes, there's a unique ID for the comic with the first "Juggernaut" appearance, but what it is?

There seem to be three main steps in finding a comic ID:

1. I'm no expert on Marvel comics, but I usually know the name of the character I'm interested in and/or the name of the comic series (e.g. "Juggernaut", *Uncanny X-men*). So the first step involves converting these character and/or series names into IDs. I've written two programs to help with this: **GetID.java** and **WhichSeries.java**. These programs utilize the "characters" and "series" tables in the database shown in Figure 2

2. I use the character/series IDs to search the "comics" table because the IDs are unique so should return better quality matches. The program for this step is called **FindComics.java**, and includes a few extra tricks (such as a year range) to narrow down the search.

3. Hopefully, step 2 will return a small enough number of comic matches that I can focus in on one comic. My **LookupComicID.java** program displays extra details about a particular comic, including its cover image.

To firm up what I mean by the above three steps, I'll run through the details of finding the two covers in Figure 1. The programs all execute from the command line, producing textual output, except for the cover images which appear in their own windows.

1. The First Appearance of the Juggernaut

Step 1 requires me to find Marvel's character ID for the Juggernaut. Using GetID.java involves:

```
> run GetID juggernaut

Result status: ok
```



```
Total matches: 1
1. name: "Juggernaut"; ID: 1009382
  thumbnail:
"http://i.annihil.us/u/prod/marvel/i/mg/5/c0/528d340442cca.jpg"
```

run.bat is a little Windows batch file that hides the call to java.exe and the setting of the classpath to use the third party libraries Commons Codec (http://commons.apache.org/proper/commons-codec/) and JSON-simple (https://code.google.com/p/json-simple/). GetID sends a suitable query over the network to the Marvel API, retrieves a response, extracts relevant information, and prints it to standard output.

The character ID for the Juggernaut is 1009382. One way of confirming this is the right Juggernaut is to paste the thumbnail image URL into a browser and have a look at the character.

Since the Juggernaut isn't a hugely important character, perhaps this is enough information for me to move to step 2. I call FindComics.java with the character ID, and leave the series ID unspecified:

```
>  run FindComics 1009382 ?
```

The '?' argument tells FindComics that no series ID is being used.

This returns the maximum of 50 matches, which is too many. So I'll restrict the search to the 1960's by including a year range argument in the FindComics call:

```
>run FindComics 1009382 ? 1960-1969

Result status: ok
Total matches: 5
Size of results array: 5
1. Title: "Uncanny X-Men (1963) #12";  issue no: 12
  On-sale date: 1965-07-10
  description: "Magneto meets his match in the menace called...the
Stranger! Can either the X-Men or the Brotherhood withstand his
power?"
  thumbnail:
"http://i.annihil.us/u/prod/marvel/i/mg/6/90/4d49e226c7d51.jpg"
  id: 12436
  page count: 36
  No. of stories: 2
    cover: "Cover #15422"
    story: "The Origin of Professor X!"

2. Title: "Uncanny X-Men (1963) #13";  issue no: 13
  On-sale date: 1965-09-10
  description: "It's always awkward to have relatives crash at your
place, especially when they're knocking down the walls! First
appearance and origin of Professor X's stepbrother, the Juggernaut!"
  thumbnail:
"http://i.annihil.us/u/prod/marvel/i/mg/9/c0/4d49e19e06f5a.jpg"
  id: 12447
  page count: 36
  No. of stories: 2
    cover: "Cover #15445"
    story: "Where Walks the Juggernaut!"

3. Title: "Uncanny X-Men (1963) #32";  issue no: 32
```



```
  On-sale date: 1967-05-10
  thumbnail:
"http://i.annihil.us/u/prod/marvel/i/mg/f/b0/4bc380feedf66.jpg"
  id: 12480
  page count: 36
  No. of stories: 2
    cover: "Cover #15513"
    story: "Beware the Juggernaut, My Son!"

4. Title: "Uncanny X-Men (1963) #33";  issue no: 33
  On-sale date: 1967-06-10
  thumbnail:
"http://i.annihil.us/u/prod/marvel/i/mg/3/30/4bc3779e11503.jpg"
  id: 12481
  page count: 36
  No. of stories: 2
    cover: "Cover #15515"
    story: "Into the Crimson Cosmos!"

5. Title: "Uncanny X-Men (1963) #46";  issue no: 46
  On-sale date: 1968-07-10
  thumbnail:
"http://i.annihil.us/u/prod/marvel/i/mg/6/60/4bc382d34829a.jpg"
  id: 12495
  page count: 36
  No. of stories: 3
    cover: "Cover #15551"
    story: "[The End of the X-Men!]"
    story: "...And Then There Were Two"
```

The matches are reported in on-sale date order, so the first match (*Uncanny X-Men* #12") is probably the one I want, even though the description mentions Magneto. It's best to move to step 3 and examine that particularly comic (ID: 12436) using LookupComicID.java:

```
> run LookupComicID 12436


Result status: ok
Total matches: 1
1. Title: "Uncanny X-Men (1963) #12";  issue no: 12
  On-sale date: 1965-07-10
  description: "Magneto meets his match in the menace called...the
Stranger! Can either the X-Men or the Brotherhood withstand his
power?"
  thumbnail:
"http://i.annihil.us/u/prod/marvel/i/mg/6/90/4d49e226c7d51.jpg"
  id: 12436
  page count: 36
  No. of stories: 2
    cover: "Cover #15422"
    story: "The Origin of Professor X!"
  No. of creators: 6
    inker: Vince Colletta
    inker: Frank Giacoia
    penciller (cover): Jack Kirby
    editor: Stan Lee
    letterer: Sam Rosen
    penciller: Alex Toth
```



```
  No. of characters: 2
    Juggernaut
    X-Men

Downloading
"http://i.annihil.us/u/prod/marvel/i/mg/6/90/4d49e226c7d51.jpg" ...
Image size: 550 x 830
Thumbnail can be saved to "image.png"
```

In addition to the extra textual output, a window is opened showing the cover (see Figure 3). The image can be saved by selecting the "Save" menu item at the top of the window.

Figure 3. The Comic Display Window for *Uncanny X-Men* #12.

I have a feeling that the comic description may be wrong, since the list of characters appearing in the comic only mentions the Juggernaut and the X-Men.

2. Wolverine versus the Hulk

Finding the cover of the first battle between the Hulk and Wolverine involves a similar three step sequence as before. First I use GetID.java to lookup the character IDs for the two heroes:

```
> run GetID wolverine

Result status: ok
Total matches: 5
Size of results array: 5
1. name: "Wolverine"; ID: 1009718
  description: "Born with super-human senses and the power to heal
from almost any wound, Wolverine was captured by a secret Canadian
```



```
organization and given an unbreakable skeleton and claws. Treated
like an animal, it took years for him to control himself. Now, he's a
premiere member of both the X-Men and the Avengers."
  thumbnail:
"http://i.annihil.us/u/prod/marvel/i/mg/c/00/528d3a1eb24ee.jpg"

2. name: "Wolverine (LEGO Marvel Super Heroes)"; ID: 1017297
  thumbnail:
"http://i.annihil.us/u/prod/marvel/i/mg/6/00/5239c3b29cb40.jpg"

  :  // 3 more matches not shown


> run GetID hulk

Result status: ok
Total matches: 9
Size of results array: 9
1. name: "Hulk"; ID: 1009351
  description: "Caught in a gamma bomb explosion while trying to save
the life of a teenager, Dr. Bruce Banner was transformed into the
incredibly powerful creature called the Hulk. An all too often
misunderstood hero, the angrier the Hulk gets, the stronger the Hulk
gets."
  thumbnail:
"http://i.annihil.us/u/prod/marvel/i/mg/c/03/526039b477c92.jpg"

2. name: "Hulk (HAS)"; ID: 1017098
  description: "The Hulk is the biggest, strongest, smashing-est hero
in the Marvel Universe - the green glue that holds his crazy family-
like team together. Hulk loves saving the world by smashing every
alien, sea creature, and planet (literally) that tries to destroy it.
He is the star of his best bud A-Bomb's web series, and just wants to
show people his good intentions!"
  thumbnail:
"http://i.annihil.us/u/prod/marvel/i/mg/d/10/5232027069e61.jpg"

   :  // 7 more matches not shown
```

These calls to GetID illustrate that using a character name when searching is a poor idea: "wolverine" matches 5 possibilities and "hulk" gives 9 hits. Part of the reason for this is that GetID searches for names beginning with the user's input, and so "hulk" also matches 'hulkling", for instance. It's more accurate to switch to the IDs for the characters (1009718 and 1009351) in subsequent searches. I chose those IDs by reading the descriptions, and by looking at the associated character thumbnail pictures.

Step 2 involves searching for comics using FindComics.java. If I call it with more than one character ID (as in this case) then it will construct a Marvel API search confined to shared appearances involving all the IDs:

```
> run FindComics "1009718,1009351" ?
```

This search looks for shared appearances of the Hulk and Wolverine, without any concern for the series. It produces too many hits so, as before, I impose an additional year range – the early 1970s:


```
> run FindComics "1009718,1009351" ? 1970-1975

Result status: ok
Total matches: 2
Size of results array: 2
1. Title: "Incredible Hulk (1962) #180";  issue no: 180
  On-sale date: 1974-10-10
  thumbnail:
"http://i.annihil.us/u/prod/marvel/i/mg/6/a0/4f567d06a4d37.jpg"
  id: 8991
  page count: 36
  No. of stories: 2
    cover: "Incredible Hulk (1962) #180"
    story: "And the Wind Howls Wendigo"

2. Title: "Incredible Hulk (1962) #182";  issue no: 182
  On-sale date: 1974-12-10
  thumbnail:
"http://i.annihil.us/u/prod/marvel/i/mg/d/20/4f5683f6ea6fc.jpg"
  id: 8993
  page count: 0
  No. of stories: 2
    cover: "Incredible Hulk (1962) #182"
    story: "Between Hammer and Anvil"
```

These results indicate a slight problem with the Marvel database – issue 181, on-sale date November 1974, is missing from the matches. Of course that's the issue I want, which I worked out by looking at the thumbnail covers for issues 180 and 182.

Although issue 181 is missing, I can try guessing its comic ID since I know the IDs for issues 180 and 182 are 8991 and 8993 respectively. I'll try looking at comic ID 8992:

```
> run LookupComicID 8992
Failed; HTTP error code: 404 Not Found
```

That means there's no comic with that ID.

Time to try another approach. I know that I want *Incredible Hulk* #181, so I'll execute a year-based search using the series ID for *Incredible Hulk*.

I return to step 1, and use GetID.java to get the series IDs for every Hulk comic:

```
> run GetID series hulk

Result status: ok
Total matches: 102
Size of results array: 50
1. title: "Hulk: Future Imperfect (1992 - 1993)"; ID: 2582
2. title: "Hulk Smash (2001)"; ID: 2581
3. title: "Hulk/Wolverine: 6 Hours (2003)"; ID: 167
4. title: "Hulk/Wolverine: 6 Hours (2003)"; ID: 2574
5. title: "Hulk: The Movie (2003)"; ID: 187

    :  // and 45 more
```


This time I included the "series" keyword when calling GetID so that it switched over to looking for a series ID instead of its default behavior of looking for character IDs.

The query generates lots of matches, but none of them are correct. The problem is that the Marvel API doesn't support searches using fragments of names, such as "hulk". The best it offers is that the text starts the title, which is why all 50 matching titles begin with the word "Hulk". Actually, I want series titles that start with "Incredible". I'll try that:

```
> run GetID series incredible

Result status: ok
Total matches: 54
Size of results array: 50
1. title: "Incredible Hulk (1962 - 1999)"; ID: 2021
  description: "Trapped in a the radioactive blast of a Gamma Bomb,
withdrawn Doctor Bruce Banner becomes the rampaging Incredible Hulk!
Witness the evolution of Marvel's most unstoppable force, from
sinister brute to uncontrollable beast and every stage in between."

2. title: "Incredible Hulk: The End (1969)"; ID: 2108

3. title: "Incredible Hulk Annual (1976 - Present)"; ID: 2983

    :  // and 47 more
```

It turns out that the title I'm after is the first match, and gives the series ID as 2021.

I've got what I wanted, but only because I remembered that the series is called *Incredible Hulk* and not *Invincible Hulk* or *Bulky Hulk*, or something. What if I can't remember the exact name of the series? That's where WhichSeries.java comes in handy. I can give it a character name, and it will query the Marvel API for all series that use that character. This is different from my previous GetID call that looked for a particular series name.

I'll employ WhichSeries to find all the series that mention the Hulk. As usual, I'll narrow down the search by using the Hulk's character ID, 1009351:

```
> run WhichSeries 1009351

Result status: ok
Total matches: 314
Size of results array: 50
1. title: "X-Men: The Complete Onslaught Epic Vol. 4 (0000 -
Present)"; ID: 3319

2. title: "What If? Classic Vol. 4 (0000 - Present)"; ID: 3314

3. title: "Marvel Masterworks: Ant-Man/Giant-Man Vol. 1 (0000 -
Present)"; ID: 3297

4. title: "Tales to Astonish (1959 - 1968)"; ID: 2080
  description: "Check out some of Marvel's most bizarre monsters as
well as the exploits of Ant-Man, The Hulk, Namor and others in this
seminal series!"

5. title: "Fantastic Four (1961 - 1998)"; ID: 2121
```



```
  description: "Witness the comic masterpiece that gave birth to the
Marvel Age! Mister Fantastic, The Thing, The Human Torch and The
Invisible Girl unite to uncover the universe's greatest mysteries and
face down would be world-conquerors like Doctor Doom, the Skrulls,
Galactus and countless others!"

6. title: "Incredible Hulk (1962 - 1999)"; ID: 2021
  description: "Trapped in a the radioactive blast of a Gamma Bomb,
withdrawn Doctor Bruce Banner becomes the rampaging Incredible Hulk!
Witness the evolution of Marvel's most unstoppable force, from
sinister brute to uncontrollable beast and every stage in between."

    :   // and another 44
```

The series are shown ordered by the year they began, and *Incredible Hulk* is the sixth match. Note that the first three matches are out of order since their start years are recorded as 0000.

I now have the series ID for the *Incredible Hulk* (ID 2021), so I'll switch back to FindComics.java and list all the comics in that series which went on-sale in 1974:

```
> run FindComics ? 2021 1974

Result status: ok
Total matches: 11
Size of results array: 11
1. Title: "Incredible Hulk (1962) #171";  issue no: 171
  On-sale date: 1974-01-10
  thumbnail:
"http://i.annihil.us/u/prod/marvel/i/mg/1/40/5284e93765e71.jpg"
  id: 8982
  page count: 0
  No. of stories: 2
    cover: "Cover #18488"
    story: "Revenge"

  :  // 8 matches not shown

9. Title: "Incredible Hulk (1962) #180";  issue no: 180
  On-sale date: 1974-10-10
  thumbnail:
"http://i.annihil.us/u/prod/marvel/i/mg/6/a0/4f567d06a4d37.jpg"
  id: 8991
  page count: 36
  No. of stories: 2
    cover: "Incredible Hulk (1962) #180"
    story: "And the Wind Howls Wendigo"

10. Title: "Incredible Hulk (1962) #181";  issue no: 181
  On-sale date: 1974-11-01
  thumbnail:
"http://i.annihil.us/u/prod/marvel/i/mg/c/70/4bb638ebb584c.jpg"
  id: 17198
  page count: 0
  No. of stories: 2
    cover: "Cover #36655"
    story: "Interior #36656"

11. Title: "Incredible Hulk (1962) #182";  issue no: 182
```



```
  On-sale date: 1974-12-10
  thumbnail:
"http://i.annihil.us/u/prod/marvel/i/mg/d/20/4f5683f6ea6fc.jpg"
  id: 8993
  page count: 0
  No. of stories: 2
    cover: "Incredible Hulk (1962) #182"
    story: "Between Hammer and Anvil"
```

The query is "`run FindComics ? 2021 1974`" – the '?' means that no character IDs are being used, 2021 is the series ID, and the date range is a single year.

Match no 10 is the November issue 181, and its comic ID is 17198, which explains why my earlier search attempt with comic ID 8992 failed. Note that only 11 comics were listed for 1974 – issue 176 isn't reported for some reason.

It would be very nice if I could search the database for a comic based on its series name and issue number (e.g. *Incredible Hulk* and 181). Unfortunately, the Marvel API doesn't support issue number searches even though the API returns that information.

Time to examine *Incredible Hulk* #181 with LookupComicID.java:

```
> run LookupComicID 17198

Result status: ok
Total matches: 1
1. Title: "Incredible Hulk (1962) #181";  issue no: 181
  On-sale date: 1974-11-01
  thumbnail:
"http://i.annihil.us/u/prod/marvel/i/mg/c/70/4bb638ebb584c.jpg"
  id: 17198
  page count: 0
  No. of stories: 2
    cover: "Cover #36655"
    story: "Interior #36656"
  No. of creators: 0
  No. of characters: 1
    Hulk

Downloading
"http://i.annihil.us/u/prod/marvel/i/mg/c/70/4bb638ebb584c.jpg" ...
Image size: 550 x 830
Thumbnail can be saved to "image.png"
```

The displayed cover is shown as Figure 4.

Figure 4. The Cover Display Window for *Incredible Hulk* #181.

The roster of characters for this comic does not mention Wolverine, which is why my initial shared appearances search for Hulk and Wolverine didn't find this issue.

A Summary of my Cover-finding Tools

The Java programs that I've implemented for finding covers, are graphically summarized in Figure 5, based on a 'slimmed-down' version of the Marvel database illustration in Figure 2.

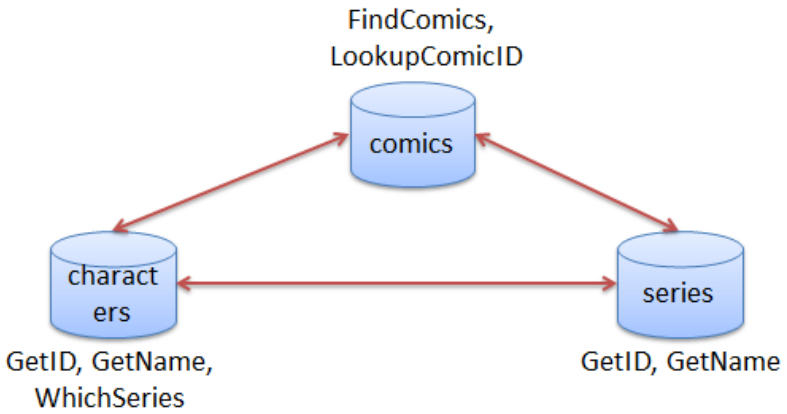


Figure 5. Tools for Finding Covers.

GetID.java can be used to map character or series names to their corresponding IDs.

I didn't utilize GetName.java in the above examples – it does the opposite of GetID by mapping a character or series ID to its corresponding name.

Figure 5 shows that my tools don't cover all of the Marvel API. In particular, there's no programs for querying the events, creators, or stories tables. Such tools could be added of course, but the existing collection is sufficient for my needs. Anyone who would like to extend the coverage is very welcome to do so.

Implementing the Cover-Finding Tools

All the programs in Figure 5 are implemented in a similar way, which breaks a Marvel API search down into three sub-tasks:

1. A URL string is created using the Marvel website address and various arguments that will be interpreted as query parameters when the URL is processed by the API at the website.

2. The URL is used to create a network link with the website, and the response to the API query is retrieved as a long string.

3. The string is converted into a JSON data structure, made up of records holding name/value pairs, collected in a results list. The benefit of employing JSON over the original string is that the data is better structured, and so can be searched more easily. Important data (i.e. on characters, series, and comics) is extracted from the JSON records and printed out.

Stages 1 and 2 utilize mostly standard Java string and network coding, but stage 3 requires a third-party library for JSON parsing. There are many such libraries – I went for a simple one, appropriately called JSON-Simple (https://code.google.com/p/json-simple/).

To illustrate the details of these three stages, I'll look at the implementation of FindComics.java, arguably the most important (and complex) of the programs listed in Figure 5.

Calling FindComics

FindComics searches Marvel's "comics" table returning matches based on user-supplied character and/or series names or IDs. It's possible to include a year range to narrow down the search. The syntax for calling the program is:

```
run FindComics (character|ID(s)|?) (series_title|ID(s)|?)
                                                  [ year[-year] ]
```

The '|" means "or", the […] means optional, and the '?'s mean "doesn't matter".

Some example calls:

- `run FindComics spider-man ? 1963`
  Find the comics containing the character "spider-man" that went on-sale in 1963. The series doesn't matter.

- `run FindComics ? "Amazing Spider-man" 1965-1967`
  Find comics in the series *Amazing Spider-man*, on-sale from the start of 1965 to the end of 1967. The characters in the series do not matter.

- `run FindComics ? 1987 1965-1967`
  Find comics in the series with ID 1987 (*Amazing Spider-Man* (1963 - 1998)) that were printed from 1965 to 1967. This is a different from the previous query which used the series name *Amazing Spider-man*. That name actually matches two different Marvel series -- "*Amazing Spider-Man* (1963 - 1998)" and "*Amazing Spider-Man* (1999 - 2013)", and so this query performs a more focused search.

- `run FindComics 1009382 "1987,454"`
  Find comics in the series "*Amazing Spider-Man* (1963 - 1998)" **and** "*Amazing Spider-Man* (1999 - 2013)" involving the character ID 1009382 (the Juggernaut). No restrictions are placed on the year of the comics. This query shows how multiple IDs can be specified, separated by ","s.

- `run FindComics "1009287,1009537" ?`
  Look for comics containing a shared appearance of Electro (ID 1009287) and the Rhino (ID 1009537); the series and year do not matter. In the case of multiple character IDs (as here), the search looks for comics that contain all the characters.

The information returned for each comic includes its ID, title, issue number, on-sale date, thumbnail URL, page count, the number of stories in the comic, and the titles of those stories (including the cover title).

The matching comics records are sorted into on-sale date order, with the earliest listed first.

The Top-level of FindComics

FindComics consists of about 50 lines inside a single main() function. The heavy lifting is delegated to static library functions defined in a separate MarvelUtils class. The first 40 or so lines of FindComics.java deal with converting a user-supplied character name into an ID, a series name into an ID, and transforming the year range string into a two-element integer array. This input manipulation is lengthy because of the different ways that a user can call FindComics (as demonstrated above).

I'll concentrate on explaining the last three lines of FindComics.java, which perform the three-stage search that I spoke about above: URL construction, network connection, response parsing. Each line in FindComics carries out one of these tasks:

```
// in FindComics.java
String urlQuery = MarvelUtils.comicsRequest(characterID, seriesID,
                          yearRange, PUBLIC_KEY, PRIVATE_KEY);
  // stage 1: construction of the URL query string

String response = MarvelUtils.sendQuery(urlQuery);
  // stage 2: contacting the Marvel website; retrieve a response

MarvelUtils.comicsResults(response, false);
  // stage 3: parse the response as a JSON data structure;
  // extract and print information
```


The next three sections will look at each stage in turn.

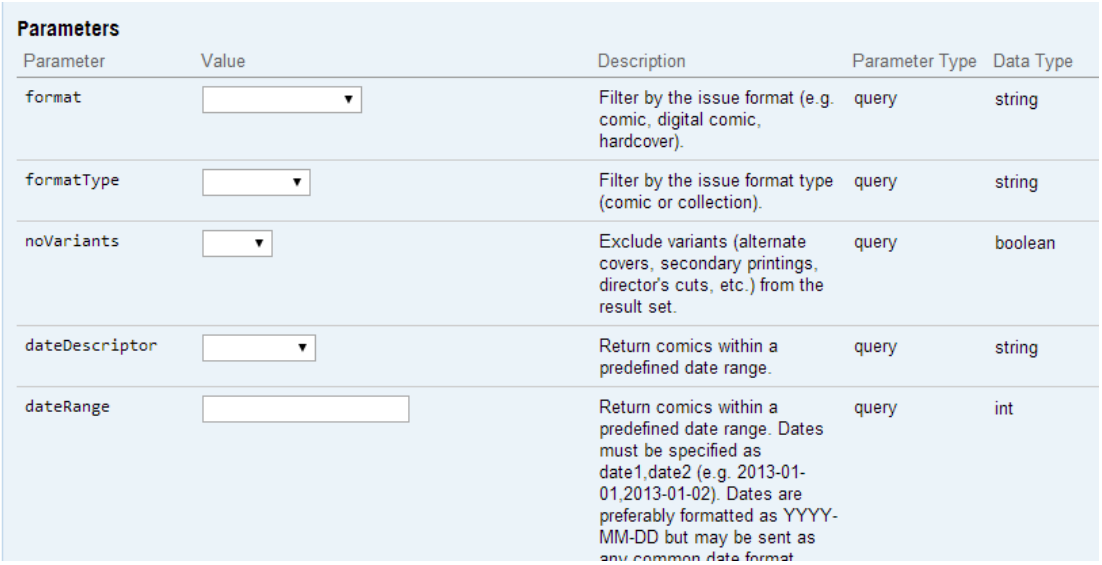
Stage 1: Constructing a Query URL

The central issue of stage 1 is selecting the parameters for encoding a particular Marvel API query as a URL. Fortunately, the Marvel Developer Portal offers an interactive test page for trying out URL queries, and viewing the results (http://developer.marvel.com/docs). Figure 6 shows part of that page, which altogether lists about 30 different kinds of URL queries.

Figure 6. Part of the Marvel Interactive Test Page.

Each row represents a basic type of query aimed at one of the six Marvel tables shown in Figure 2. Figure 6 shows six ways of accessing the "characters" table, and five "comics" queries. Each row includes a helpful one-line summary of what the query can do, and it's fairly apparent that I should employ a "GET /v1/public/comics" query to retrieve a list of comics.

Clicking on a row, expands the field to display documentation on the data formats returned by a response, and an interactive form for entering parameters and executing a query (see Figure 7).



Parameters

Parameter	Value	Description	Parameter Type	Data Type
format		Filter by the issue format (e.g. comic, digital comic, hardcover).	query	string
formatType		Filter by the issue format type (comic or collection).	query	string
noVariants		Exclude variants (alternate covers, secondary printings, director's cuts, etc.) from the result set.	query	boolean
dateDescriptor		Return comics within a predefined date range.	query	string
dateRange		Return comics within a predefined date range. Dates must be specified as date1,date2 (e.g. 2013-01-01,2013-01-02). Dates are preferably formatted as YYYY-MM-DD but may be sent as	query	int

Figure 7. Some Parameter Fields for the "GET /v1/public/comics" Query Form.

The FindComics.java program described earlier utilizes character and series names/IDs and a year range to search for comic records. Inside FindComics, character and series names are converted to IDs, so the actual query sent to Marvel may utilize three kinds of data: character ID(s), series ID(s), and a year range. FindComics passes this input to a MarvelUtils.comicsRequest() method which builds the query. The method's signature is:

```
public static String comicsRequest(
            String charId, String seriesId, int[] yearRange,
            String publicKey, String privateKey)
```

The charID string may contain a single character ID, or perhaps several separated by a comma (e.g. "1009287,1009537"), or be null if no character ID is required in the search. The seriesID argument has a similar format, and the yearRange[] array may contain a single year integer, two integers for a range, or be null.

The public and private key arguments of comicsRequest() are required in URL queries sent from standalone applications (such as FindComics.java). However, the http://developer.marvel.com/docs test page doesn't need them to be explicitly added as parameters since it 'knows' who I am after I logged into the Marvel site.

This all means that I need to modify at most three parameter fields in the "GET /v1/public/comics" query form – type 0 or more character IDs into the "characters" (or "sharedAppearances") field, enter 0 or more series IDs into the "series" field, and write an optional year range into the "dateRange" field.

If I'm only employing a single character ID, then I enter it into the "characters" field. However, if the search is utilizing two characters then a better choice is the "sharedAppearances" field, as shown in Figure 8.

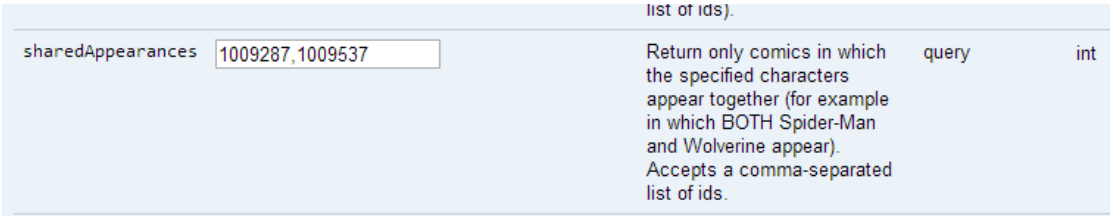


Figure 8. Two Characters IDs in a Shared Appearance.

When I press the "Try It Out" button at the bottom of the form, the URL query is constructed, and shown on-screen. It is delivered to the Marvel website, and the response string is eventually displayed, as in Figure 9.

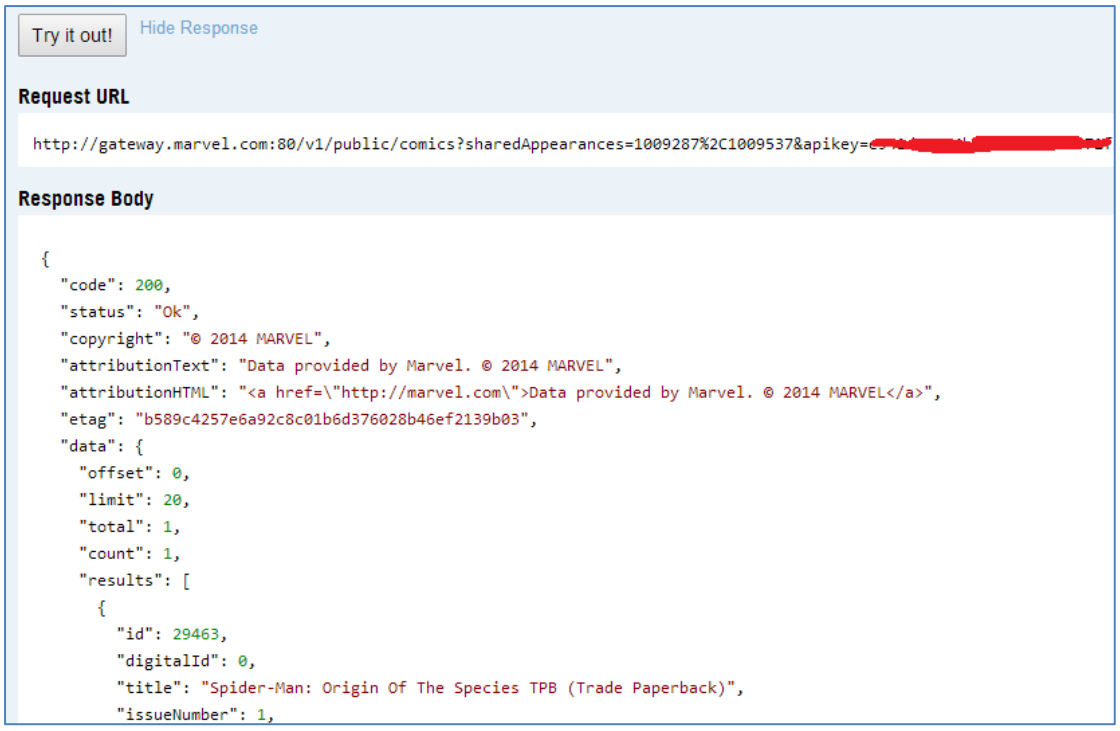


Figure 9. URL Query and Response.

The URL query shown in Figure 9 is:

```
http://gateway.marvel.com:80/v1/public/comics?
          sharedAppearances=1009287%2C1009537&apikey=XXXXX
```

The response string at the bottom of Figure 9 is a JSON record which I'll explain when I talk about the coding of stage 3.

The URL for accessing the "comics" table is `"http://gateway.marvel.com:80/v1/public/comics?"`, while the parameters part has the format:

```
field1=value1&field2=value2&field3=value3...
```

The above query uses two parameters, "`sharedAppearances`" and "`apikey`". Each parameter has the format field=value, and are separated by ampersands. A field name and value comes from the corresponding webpage form field, with the value string URL-encoded. This means that characters with special meaning in URLs, such as '/' and ' ' are represented as hexadecimals. For instance, in the query above, ',' has been replaced by "%2C".

The "sharedAppearances" parameter ("`sharedAppearances=1009287%2C1009537`") comes from Figure 8. The "apikey" field value is my public API key, and was added automatically by the webpage when the URL was constructed.

A few more tests with different mixes of character IDs, series IDs, and year ranges allowed me to implement the necessary string building code inside MarvelUtils.comicsRequest():

```
// globals
private static final String MARVEL_ADDR =
                  "http://gateway.marvel.com/v1/public/";
    // address of Marvel's developer portal


public static String comicsRequest(
          String charId, String seriesId, int[] yearRange,
          String publicKey, String privateKey)
{
  if ((charId == null) && (seriesId == null) &&
      (yearRange == null)) {
    System.out.printf("Can not construct a query with no data");
    return null;
  }

  int[] ids = getIDs(charId);
  boolean isShared = false;
  if ((ids != null) && (ids.length > 1))  // shared appearance
    isShared = true;

  Long timeStamp = new Date().getTime();
  String hash = genMD5hash(timeStamp + privateKey + publicKey);

  String urlStr = MARVEL_ADDR + "comics?" +
      "limit=50" +
      ((charId == null) ? "" : (
         (isShared ? ("&sharedAppearances=" + URLencode(charId)) :
                     ("&characters=" + URLencode(charId)))
       )) +
      ((seriesId == null) ? "" :
                   ("&series=" + URLencode(seriesId)) ) +
      ((yearRange == null) ? "" :
                   ("&dateRange=" + buildDateRange(yearRange)) ) +
      "&orderBy=onsaleDate" +
      "&ts=" + timeStamp +
      "&apikey=" + publicKey +
      "&hash=" + hash;

  return urlStr;
}  // end of comicsRequest()
```

The construction of the URL string begins on the line:


```
    String urlStr = MARVEL_ADDR + "comics?" + ...
```

The choice of whether to use a "sharedAppearances" or a "characters" parameter depends on the value of the isShared boolean which is set by examining the number of character IDs returned by MarvelUtil.getIDs():

```
public static int[] getIDs(String s)
// convert IDs string into an array of ID integers
{
  if (!isID(s))
    return null;
  else {
    String[] idStrs = s.split(",");
    int[] ids = new int[idStrs.length];
    for (int i=0; i < idStrs.length; i++)
      ids[i] = toInt(idStrs[i]);
    return ids;
  }
}  // end of getIDs()
```

Another tricky field to fill is "dateRange" because it uses a YYYY-MM-DD format (e.g. 1967-12-01) whereas my FindComics only supplies years. The solution in buildDateRange() is to add suitable month and year substrings. For instance, the year range 1967-1968 becomes the string "1967-01-01,1968-12-31" (except that I use the URL-encoding of ',', which is "%2C").

Other parameters added by comicsRequest() are "limit", "orderBy", "ts", "apikey", and "hash". "limit" is set to 50 to limit the number of matches to a maximum of 50, while "orderBy" is assigned "onsaleDate" so that the comic records are sorted into increasing on-sale date.

Queries sent from standalone applications, as in the case of my cover-finding tools, must include the "ts", "apikey", and "hash" parameters. "ts" must be assigned a unique timestamp, for which I utilize `new Date().getTime()` to supply the number of milliseconds since January 1, 1970. The "apikey" field is assigned my public key. The "hash" field must be a md5 digest of the timestamp parameter, my private key, and public key (e.g. md5(ts+privateKey+publicKey), which is created by the genMDhash() method:

```
private static String genMD5hash(String msg)
// transforms message string to a MD5 hash string
{
  String hash = null;
  try {
    MessageDigest md = MessageDigest.getInstance("MD5");

    byte[] digestBytes = md.digest( msg.getBytes("UTF-8") );

    hash = new String( Hex.encodeHex(digestBytes) );
               // Hex class is from org.apache.commons.codec.binary
  }
  catch (NoSuchAlgorithmException e) {
    System.out.println("No MD5 algorithm found");
    System.exit(1);
  }
  catch (UnsupportedEncodingException e) {
```



```
    System.out.println("No Hex encoder found");
    System.exit(1);
  }
  return hash;
}  // end of genMD5hash()
```

Although Java's standard libraries include a MessageDigest class, they don't have a library for converting the bytes output from the digest function into a hexadecimal string suitable for a URL parameter. I use the Commons Codec library (http://commons.apache.org/proper/commons-codec/) to fill the gap.

Although this section has focused on the comicsRequest() method, the MarvelUtils class contains several other URL-building methods (e.g. characterRequest(), seriesRequest(), and comicIDRequest()). They all have a similar coding structure as comicsRequest(), and reuse many of the same methods, such as genMD5hash(). As a consequence, I won't explain them here. They're fully documented inside the MarvelUtils class.

Stage 2: Contacting the Website

At the end of stage 1, a URL in string form is ready to use. The query format inside the URL means that it must be sent as a GET request to the Marvel website. In addition, my code must read back the response. These tasks are done by MarvelUtils.sendQuery():

```
public static String sendQuery(String urlStr)
// contact the specified URL, and return the response as a string
{
  String response = null;
  HttpURLConnection conn = null;
  try {
    URL url = new URL(urlStr);

    conn = (HttpURLConnection) url.openConnection();
    conn.setRequestMethod("GET");
    conn.setRequestProperty("Accept", "application/json");

    if (conn.getResponseCode() != 200) {
      System.out.println("Failed; HTTP error code: " +
                  conn.getResponseCode() +
                  " " + conn.getResponseMessage());
      return null;
    }

    BufferedReader br = new BufferedReader(
              new InputStreamReader((conn.getInputStream())));
    String line = null;
    StringBuilder sb = new StringBuilder();

    while ((line = br.readLine()) != null)
      sb.append(line);
    br.close();

    response = sb.toString();
```



```
    }
    catch (MalformedURLException e) {
      System.out.println("Error: Not a valid URL");
    }
    catch (IOException e) {
      System.out.println("Error: Could not retrieve search results");
    }
    finally {
      conn.disconnect();
    }
    return response;
  }  // end of sendQuery()
```

The code is made a little more complicated by having to handle various kinds of error, such as an error response or an incorrect URL. Also, the GET's request property is set to accept "application/json" so that the response will be parsable as JSON in stage 3.

Stage 3: Parsing the Response as a JSON Data Structure

JSON (JavaScript Object Notation) is human-readable text suitable for transmitting data objects consisting of attribute-value pairs (http://en.wikipedia.org/wiki/JSON). There are several third-party JSON libraries for Java, with JSON-simple (https://code.google.com/p/json-simple/) being one of the simpler, but more than adequate for the straightforward JSON structures used by the Marvel API.

A JSON record is a set of key-value pairs which JSON-simple converts into a JSONObject instance. JSONObject is a Java Map, allowing the data pairs to be easily accessed via their keys. A list of JSON structures (such as a collection of JSON records about comics) is mapped to a JSONArray instance, which implements the Java List interface, and so offers simple ways to search and traverse the list

The JSON structure returned by the Marvel API is fairly easy to analyze since the interactive query form displays the response to a query (see Figure 9). All responses are encoded as a JSON record which looks like:

```
{
  "code": 200,
  "status": "Ok",
  "etag": "....",
  "data": {
    "offset": 0,
    "limit": 50,
    "total": 101,
    "count": 50,
    "results": [ {...} {...} {...} ... ]
 }}
```

A JSON list is shown in […] brackets, and each JSON record is wrapped in {…}s.

The "code" and "status" fields report whether the query was accepted and processed. The "total" field states the number of matching records extracted from the table being queried. However, there's an upper-bound on the number returned to the user based on the "limit" value (which in many of my queries is set to 50). The number of records

returned in the response is shown in the "count" field, and the data is stored as a list of JSON records in the "results" field.

The first task of stage 3 is to convert the response string into a JSONObject instance, by calling JSONParser.parse() inside MarvelUtils.extractResults():

```
private static JSONArray extractResults(String jsonText)
{
  if (jsonText == null)
    return null;

  JSONArray results = null;
  JSONParser parser = new JSONParser();
  try{
    JSONObject jo = (JSONObject)parser.parse(jsonText);
         // response string becomes a JSON object

    // look inside the JSON object for the results array
    String status = ((String)jo.get("status")).toLowerCase();
    System.out.println("\nResult status: " + status);
    if (!status.equals("ok"))
      return null;

    JSONObject jod = (JSONObject)jo.get("data");

    System.out.println("Total matches: " + jod.get("total"));
    results = (JSONArray)jod.get("results");
  }
  catch(Exception e)
  {  System.out.println("Could not extract results: " + e);  }

  return results;
} // end of extractResults()
```

extractResults() prints the "status" and "total" values, and returns the "results" list as a JSONArray object.

extractResults() is called by every response-processing method implemented in MarvelUtils, but the results JSONArray is processed in different ways depending on the kind of records in the list. For instance, MarvelUtils.comicsResults(), which processes a list of comics results, begins like so:

```
public static String comicsResults(String queryResponse,
                                           boolean showFull)
{
  String thumbAddress = null;
  JSONArray results = extractResults(queryResponse);
  if (results == null)
    return null;

  try{
    if (results.size() != 1)
      System.out.println("Size of results array: " +
                                           results.size());
    for(int i=0; i < results.size(); i++) {
      JSONObject result = (JSONObject)results.get(i);

      // print out the the results in the i-th record...
```



```
      // the details will be explained in a moment...
    }
  }
  catch(Exception e)
  {  System.out.println(e);  }

  return thumbAddress;
}  // end of comicsResults()
```

The length of the results JSONArray is printed, and then a loop iterates through the list examining each JSON comic record as a JSONObject instance.

The format of a JSON comic record is explained in the documentation that precedes the interactive form for sending a query at http://developer.marvel.com/docs. Naturally, it is rather lengthy, since there are many kinds of data stored for each comic; a fragment of the information is shown in Figure 10.

Comic {

id (int, *optional*): The unique ID of the comic resource.,
digitalId (int, *optional*): The ID of the digital comic representation of this comic. Will be 0 if th
title (string, *optional*): The canonical title of the comic.,
issueNumber (double, *optional*): The number of the issue in the series (will generally be 0 for
variantDescription (string, *optional*): If the issue is a variant (e.g. an alternate cover, second
description (string, *optional*): The preferred description of the comic.,
modified (Date, *optional*): The date the resource was most recently modified.,
isbn (string, *optional*): The ISBN for the comic (generally only populated for collection formats
upc (string, *optional*): The UPC barcode number for the comic (generally only populated for p
diamondCode (string, *optional*): The Diamond code for the comic.,
ean (string, *optional*): The EAN barcode for the comic.,
issn (string, *optional*): The ISSN barcode for the comic.,
format (string, *optional*): The publication format of the comic e.g. comic, hardcover, trade pap
pageCount (int, *optional*): The number of story pages in the comic.,
textObjects (Array[TextObject], *optional*): A set of descriptive text blurbs for the comic.,

Figure 10. Part of the Documentation for a Comic Record.

Each of the record fields is translated into a key-value pair in the JSONObject, and so can be accessed with a Map.get() call. For example, to access the ID and title of a record would require:

```
JSONObject result = (JSONObject)results.get(i);
            // get the i-th record in the results list
String id = result.get("id");
String title = result.get("title");
```

Note that the default type for returned values is String, and so the "id" field value would need to be explicitly converted into an integer.

Data which is labeled as optional in the Marvel documentation in Figure 10 may not be returned in a Marvel API response. In that case, a Map.get() call using that key would return null.

comicResults() is called with a boolean flag showFull which controls how much information is printed. If showFull is false, then the title, issue no., on-sale date, description, thumbnail, id, page count, and story details are reported for each record. When showFull is true, creator and character information is also printed for each comic. This is implemented inside the comicsResults() method by:

```
// inside comicResults()
// code for printing the i-th record
          :
JSONObject result = (JSONObject)results.get(i);

System.out.println((i+1) + ". Title: \"" +
              result.get("title") + "\";  issue no: " +
              result.get("issueNumber"));
reportSaleDate(result);

String description = (String)result.get("description");
if ((description != null) && (description.length() > 0))
  System.out.println("  description: \"" +  description + "\"");

JSONObject thumbnail = (JSONObject)result.get("thumbnail");
if (thumbnail != null) {
  thumbAddress = thumbnail.get("path") +
                      "." + thumbnail.get("extension");
  System.out.println("  thumbnail: \"" +  thumbAddress + "\"");
}

System.out.println("  id: " + result.get("id"));
System.out.println("  page count: " +  result.get("pageCount"));

reportStories( (JSONObject)result.get("stories") );

if (showFull) {
  reportCreators( (JSONObject)result.get("creators") );
  reportCharacters( (JSONObject)result.get("characters") );
}
       :
```

The printing of the thumbnail image URL illustrates how more complex data is encoded in JSON-simple. In the Marvel documentation, the thumbnail field is explained like so:

> **thumbnail** (Image, *optional*): The representative image for this comic.,

The implementer must then look at the documentation for an "Image" record to find out about its internal structure:

> **Image {**
>
> **path** (string, *optional*): The directory path of to the image.,
>
> **extension** (string, *optional*): The file extension for the image.
>
> **}**

In terms of JSON-simple, this means that result.get("thumbnail") will return a JSONObject instance, which will contain two fields named "path" and "extension" (e.g. "http://i.annihil.us/u/prod/marvel/i/mg/a/20/4d9f7e73c22f7" and "jpg"). My comicResults() method appends these two strings together, to create a valid URL pointing at the thumbnail image at Marvel's website.

There are two other stage 3 methods in MarvelUtils: characterResults() and seriesResults(). characterResults() is used to print the JSON results for a list of matching character records, and seriesResults() deals with a list of series records. Although the details of these methods are a little different from comicResults(), they all use extractResults() to get a results JSONArray, and then loop through the records using Map.get() to access interesting fields. Consequently, I won't explain them here, but the code is fully documented inside the MarvelUtils class.

