DevDays HL7® FHIR®

HL7 FHIR

HAPI FHIR Introduction

James Agnew, Smile CDR

HL7 FHIR DevDays 2020, Virtual Edition US, June 15–18, 2020 | @HL7 @FirelyTeam | #fhirdevdays | www.devdays.com/us

About Me

- **Name:** James Agnew
- **Background:**
  - Project Lead @ HAPI
  - Head Geek and CTO @ Smile CDR

Sharing is Caring: http://bit.ly/2qxO7si

- This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

- All code samples are available on GitHub in fully working form

Today's Objectives

- Whirlwind tour of HAPI FHIR:
  - Parser
  - Client
  - Server (Facade)
  - JPA Server (Storage Engine)
- Introduce exercises for each

Tomorrow's Objectives
2:45 - 3:25

- Interceptors introduction
- Security basics in HAPI FHIR
- Partitioning and Multitenancy
- Let's build exercises for these

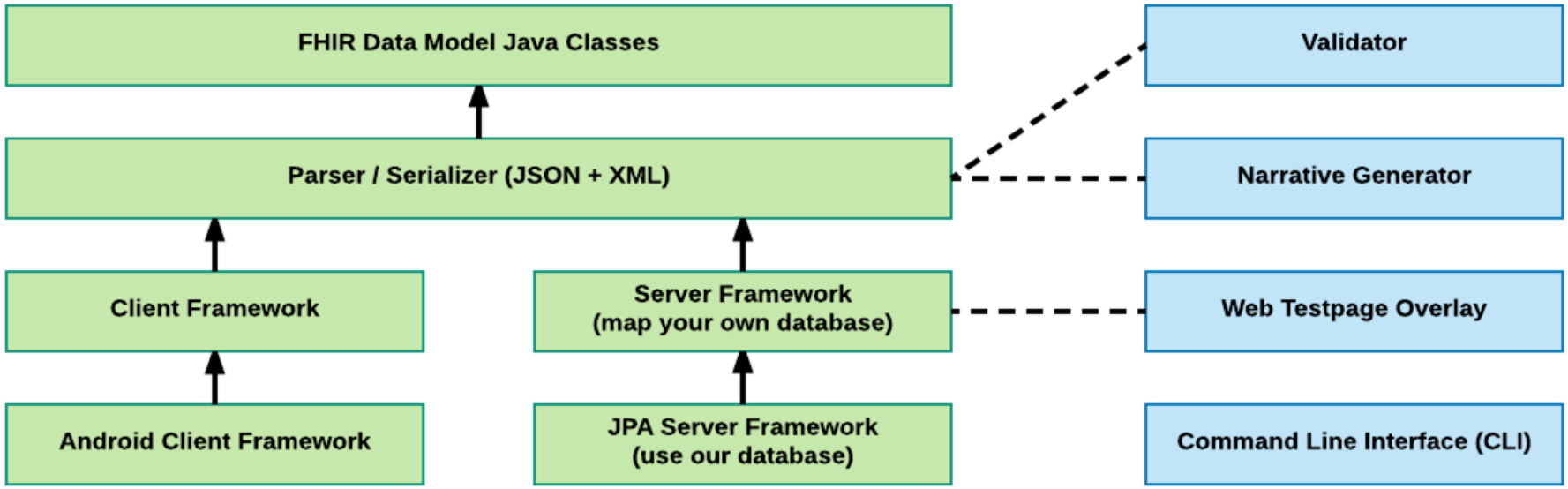
Office Hours
4:15 - 4:45 Tuesday

HAPI FHIR

- HAPI started in 2001 as an HL7 v2 Library
- HAPI FHIR was first released in 2013
- Designed as a modular set of FHIR components
- Licensed ASL 2.0

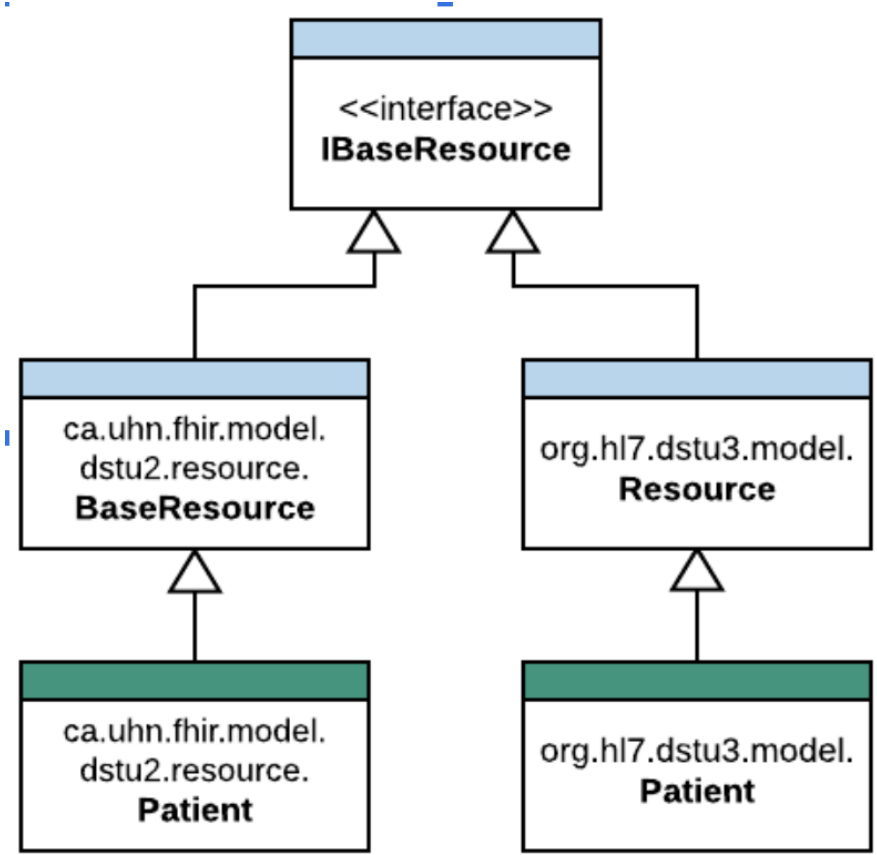
```
HL7 v2    - http://hapifhir.github.com/hapi-hl7v2
HL7 FHIR  - http://hapifhir.io
```


Modules in the Project



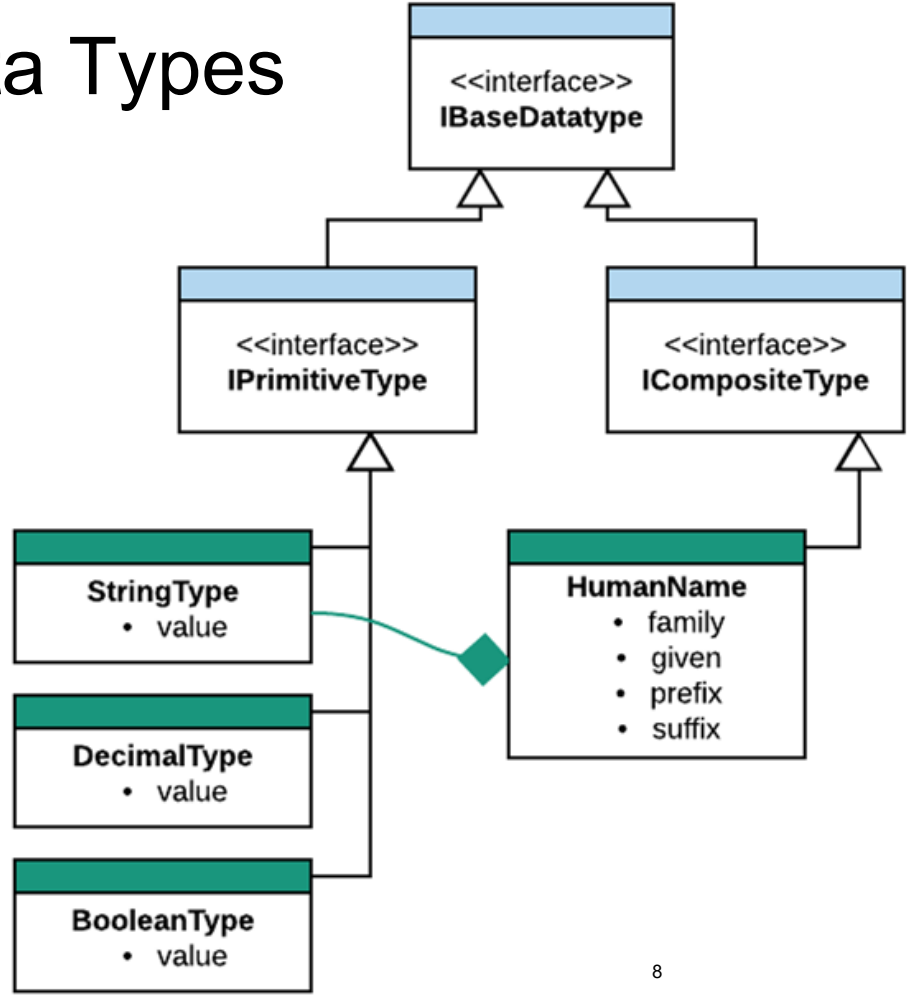
Structure Classes: Resources

- HAPI Defines several sets of classes which form the data model
- Resource definition classes implement `IBaseResource`
- Examples: Patient, CarePlan, Encounter, Practitioner, Medication



Structure Classes: Data Types

- HAPI also defines a class for each data type
- Primitive classes are named `[name]Type`
- Primitive types include: StringType, BooleanType
- Composite types include: Address, Ratio, HumanName



Creating A Resource

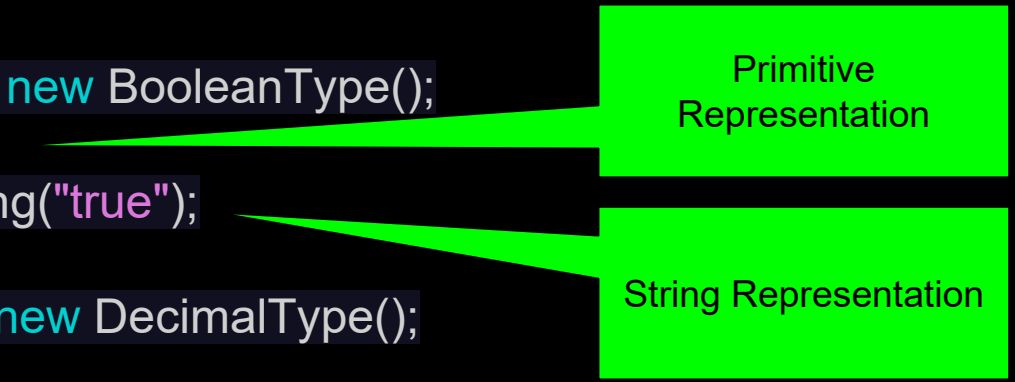
```
public class Example01_CreateAPatient {
  public static void main(String[] theArgs) {

    // Create a resource instance
    Patient pat = new Patient();

    // Add a "name" element
    HumanName name = pat.addName();
    name.setFamily("Simpson").addGiven("Homer").addGiven("J");

    // Add an "identifier" element
    Identifier identifier = pat.addIdentifier();
    identifier.setSystem("http://acme.org/MRNs").setValue("7000135");

    // Model is designed to be chained
    pat.addIdentifier().setSystem("http://acme.org/MRNs").setValue("12345");
}}
```

Enumerated Types

```
public class Example02_EnumeratedTypes {
  public static void main(String[] theArgs) {

    Patient pat = new Patient();

    pat.addName().setFamily("Simpson").addGiven("Homer").addGiven("J");
    pat.addIdentifier().setSystem("http://acme.org/MRNs").setValue("7000135");

    // Enumerated types are provided for many coded elements
    ContactPoint contact = pat.addTelecom();
    contact.setUse(ContactPoint.ContactPointUse.HOME);
    contact.setSystem(ContactPoint.ContactPointSystem.PHONE);
    contact.setValue("1 (416) 340-4800");

    pat.setGender(Enumerations.AdministrativeGender.MALE);
  }}
```

Primitive Datatypes

```
DateTimeType effective = new DateTimeType();
effective.setValue(new Date());
effective.setValue(new Date(), TemporalPrecisionEnum.MINUTE);
effective.setValueAsString("2017-09-11T14:35:00-07:00");

BooleanType active = new BooleanType();
active.setValue(true);
active.setValueAsString("true");

DecimalType value = new DecimalType();
value.setValue(1.2d);
value.setValueAsString("1.20000");
```



The HAPI FHIR Context

- The starting point for much of the HAPI-FHIR API is the `FhirContext` class
- FhirContext acts as a factory for the rest of the API, including the two parsers:
  - XmlParser
  - JsonParser
- FhirContext is designed to be created once and reused (important for performance!)

Encoding A Resource

```
public class Example04_EncodeResource {
 public static void main(String[] theArgs) {

   // Create a Patient
   Patient pat = new Patient();
   pat.addName().addFamily("Simpson").addGiven("Homer").addGiven("J");
   pat.addIdentifier().setSystem("http://acme.org/MRNs").setValue("7000135");
   pat.addTelecom().setUse(ContactPointUse.HOME).setSystem(ContactPointSystem.PHONE).setValue("1 (416) 340-4800");
   pat.setGender(AdministrativeGender.MALE);

   // Create a context
   FhirContext ctx = FhirContext.forDstu3();

   // Create a JSON parser
   IParser parser = ctx.newJsonParser();
   parser.setPrettyPrint(true);

   String encode = parser.encodeResourceToString(pat);
   System.out.println(encode);

 }
}
```

Encoding A Resource

```
public class Example04_EncodeResource {
 public static void main(String[] theArgs) {

   // Create a Patient
   Patient pat = new Patient();
   pat.addName().addFamily("Simpson").addGiven("Homer").addGiven("J");
   pat.addIdentifier().setSystem("http://acme.org/MRNs").setValue("7000135");
   pat.addTelecom().setUse(ContactPointUse.HOME).setSystem(ContactPointS
   pat.setGender(AdministrativeGender.MALE);

   // Create a context
   FhirContext ctx = FhirContext.forDstu3();

   // Create a JSON parser
   IParser parser = ctx.newJsonParser();
   parser.setPrettyPrint(true);

   String encode = parser.encodeResourceToString(pat);
   System.out.println(encode);

 }
}
```

```
{
  "resourceType": "Patient",
  "identifier": [
                {
                "system": "http://acme.org/MRNs",
                "value": "7000135"
                }
  ],
  "name": [
                {
                "family": "Simpson",
                "given": [
                "Homer",
                "J"
                ]
                }
  ],
  "telecom": [
                {
                "system": "phone",
                "value": "1 (416) 340-4800",
                "use": "home"
                }
  ],
  "gender": "male"
}
```

Clients:
Two Distinct Flavours in HAPI FHIR

Annotation

```
public interface SampleClient extends IRestfulClient {

  @Create
  MethodOutcome create(@ResourceParam Patient
thePatient);

  @Read
  Patient read(@IdParam IdType theId);

}
```

- You create an annotated interface for your specific needs
- Similar to JAX-RS or Spring REST (but does not use these frameworks)

Generic/Fluent

```
MethodOutcome outcome = client
    .create()
    .resource(pat)
    .execute();
```

- Use chained method calls to do anything you need
- This is generally easier and more popular

Create a Patient

```
public class Example06_ClientCreate {
 public static void main(String[] theArgs) {
   Patient pat = new Patient(); // <-- Not shown: Populate the patient

   // Create a context
   FhirContext ctx = FhirContext.forR4();

   // Create a client
   String serverBaseUrl = "http://fhirtest.uhn.ca/baseR4";
   IGenericClient client = ctx.newRestfulGenericClient(serverBaseUrl);

   // Use the client to store a new resource instance
   MethodOutcome outcome = client
         .create()
         .resource(pat)
         .execute();

   // Print the ID of the newly created resource
   System.out.println(outcome.getId());
```

Create a Patient

```
public class Example06_ClientCreate {
 public static void main(String[] theArgs) {
   Patient pat = new Patient(); // <-- Not shown: Populate the patient

   // Create a context
   FhirContext ctx = FhirContext.forR4();

   // Create a client
   String serverBaseUrl = "http://hapi.fhir.org/baseR4";
   IGenericClient client = ctx.newRestfulGenericClient(serverBaseUrl);

   // Use the client to store a new resource instance
   MethodOutcome outcome = client
         .create()
         .resource(pat)
         .execute();

   // Print the ID of the newly created resource
   System.out.println(outcome.getId());
```

http://hapi.fhir.org/baseR4/Patient/12345/_history/1

Create a Patient

```
public class Example06_ClientCreate {
 public static void main(String[] theArgs) {
   Patient pat = new Patient(); // <-- Not shown:

   // Create a context
   FhirContext ctx = FhirContext.forR4();

   // Create a client
   String serverBaseUrl = "http://hapi.fhir
   IGenericClient client = ctx.newR
                                GenericClient(serverBaseUrl);

   // Use the client to store a new resource instance
   MethodOutcome outcome = client
         .create()
         .resource(pat)
         .execute();

   // Print the ID of the newly created resource
   System.out.println(outcome.getId());
```

MethodOutcome holds any information returned by the server *POSSIBLY* including:

- **Resource ID and Location** (from Location header)
- **Resource Body** (from response body)
- **OperationOutcome** (from response body)

Searching (3)

```
public class Example08_ClientSearch {
 public static void main(String[] theArgs) {
   FhirContext ctx = FhirContext.forDstu3();
   IGenericClient client = ctx.newRestfulGenericClient("http://fhirtest.uhn.ca/baseDstu3");

   // Build a search and execute it
   Bundle response = client.search()
       .forResource(Patient.class)
       .where(Patient.NAME.matches().value("Test"))
       .and(Patient.BIRTHDATE.before().day("2014-01-01"))
       .count(100)
       .returnBundle(Bundle.class)
       .execute();

   // How many resources did we find?
   System.out.println("Responses: " + response.getTotal());

   // Print the ID of the first one
   System.out.println("First response ID: " + response.getEntry().get(0).getResource().getId());
```

Other Operations

- Lots of other FHIR operations are supported in the client:
  - History, CapabilityStatement, Transactions, Delete, etc.

- There is also lots of support for advanced features:
  - Version aware updates, Get-if-newer, etc.

Reminder about docs:

http://hapifhir.io/doc_rest_client.html

Servers

Different Server Modules in HAPI FHIR

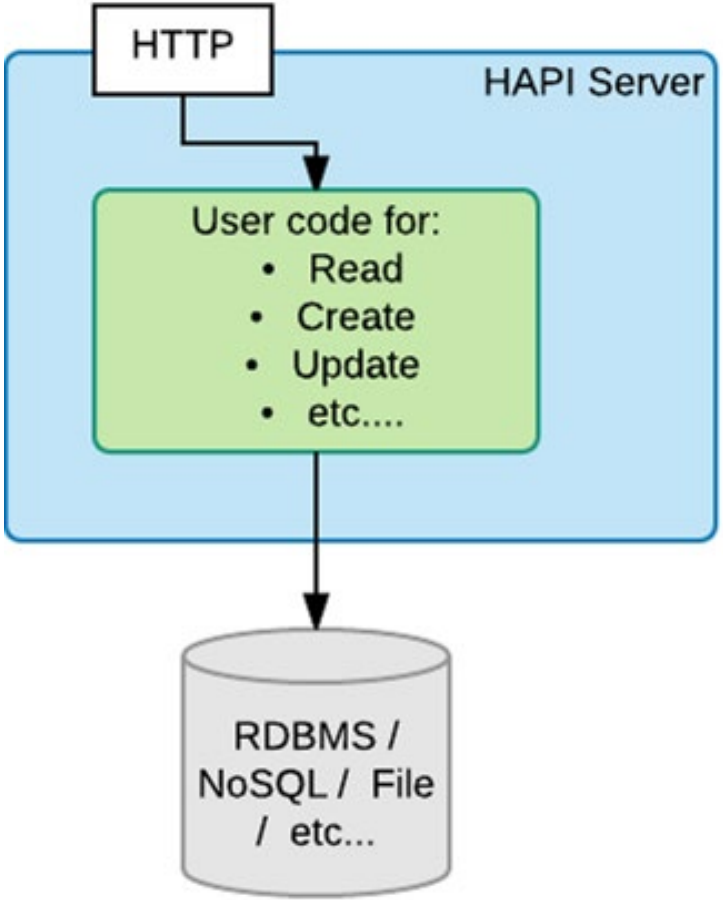
- **Plain Server**: You bring the storage, HAPI FHIR brings the REST
  - Often called a **Facade Layer**
- **JPA Server**: HAPI FHIR brings storage and REST
  - Uses its own database schema and needs an RDBMS.
- **JAX-RS**: Like the plain server but based on JAX-RS
  - This is not as full-featured. Use only if you absolutely need JAX-RS. **Not covered here!**

HAPI FHIR Plain Server

- HAPI FHIR provides a REST Server framework
- Based on standard JEE/Servlet 2.5+ (Tomcat, Glassfish, Websphere, JBoss, etc)
- Inspired by (but not based on) JAX-RS, RestEasy, Spring REST, etc.

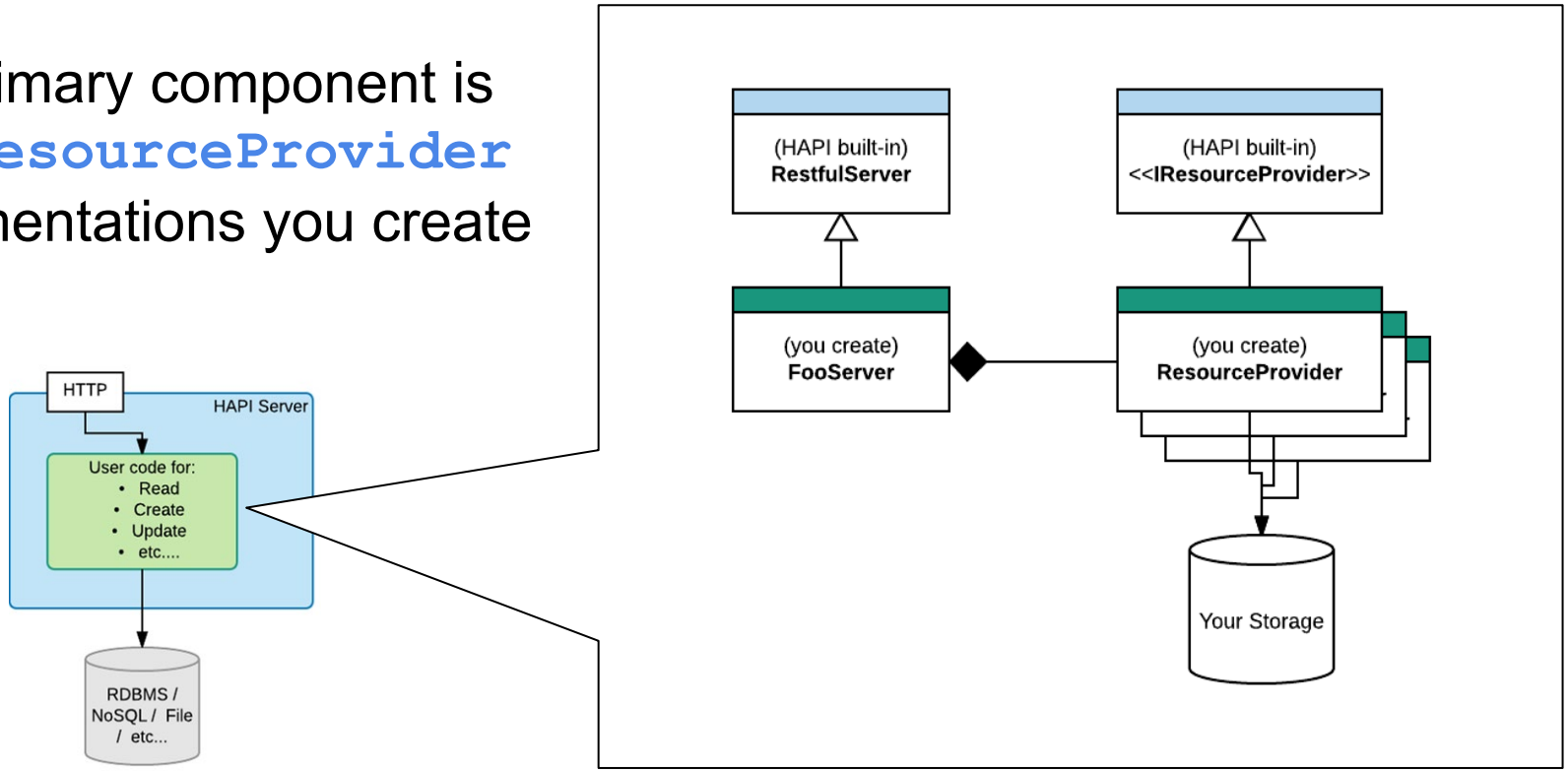
Creating a Server: Resource Providers

- You supply Java code for CRUD operations you want to support in your server
  - Read
  - Create
  - Update
  - Delete
  - Search
  - etc…



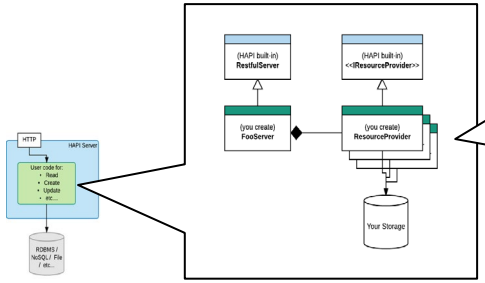
Creating a Server: Resource Providers

The primary component is the `IResourceProvider` implementations you create



Creating a Server: Resource Providers

- Your code can be deployed in a Java Servlet Container



Apache Tomcat

jetty://

WildFly

your-server.war

FooServer.class

ResourceProvider.class

Creating Resource Providers

- ResourceProviders implement the `IResourceProvider` interface and the `getResourceType()` method
- You create one resource provider **per resource type**

```
public class Example01_StubResourceProvider implements IResourceProvider {

   public Class<Patient> getResourceType() {
      return Patient.class;
   }

   // your code to handle resources

}
```

An Example Resource Provider

```
public class Example01_StubResourceProvider implements IResourceProvider {
   public Class<? extends IBaseResource> getResourceType() {  return Patient.class;  }

   @Read
   public Patient read(@IdParam IdType theId) {
      return null; // populate this
   }

  @Create
   void create(@ResourceParam Patient thePatient) {
      // save the resource
   }

   @Search
   List<Patient> search(
      @OptionalParam(name="family") StringParam theFamily,
      @OptionalParam(name="given") StringParam theGiven
      ) {
        return null; // populate this
```

An Example Resource Provider

```
public class Example01_StubResourceProvider implements IResourceProvider {
   public Class<? extends IBaseResource> getResourceType() {  return Patient.class;  }

   @Read
   public Patient read(@IdParam IdType theId) {
      return null; // populate this
   }

   @Create
   void create(@ResourceParam Patient thePatient) {
      // save the resource
   }

   @Search
   List<Patient> search(
      @OptionalParam(name="family") StringParam theFamily,
      @OptionalParam(name="given") StringParam theGiven
      ) {
        return null; // populate this
```

Read Method (HTTP GET)

Create Method (HTTP POST)

Search Method (HTTP GET)

Adding More Operations

- There are @Annotations for many more FHIR operations and features
- See the docs to see many examples:

  https://hapifhir.io/hapi-fhir/docs/server_plain/rest_operations.html

Now… Let's Build!
http://bit.ly/2qxO7si

Client:

https://github.com/FirelyTeam/fhirstarters/tree/master/java/hapi-fhirstarters-client-skeleton - http://bit.ly/2XsWOAr

Server:

https://github.com/FirelyTeam/fhirstarters/tree/master/java/hapi-fhirstarters-simple-server - http://bit.ly/2OrzIWJ

JPA:

https://github.com/hapifhir/hapi-fhir-jpaserver-starter - http://bit.ly/hapi-fhir-jpa

Bonus: Release Cycle

- HAPI FHIR releases four times per year
- See the changelog to see what's new: https://hapifhir.io/hapi-fhir/docs/introduction/changelog.html
- See the versions page to learn about supported FHIR versions https://hapifhir.io/hapi-fhir/docs/introduction/versions.html