Home Health Prospective Payment System

Java code design

PBL-047

Table of Contents

Home Health Prospective Payment System Java code design

In 2008, 3M was tasked with providing a new version of the Home Health Prospective Payment System (HH-PPS) developed using Java and providing public source code. This change replaces the previous pseudo code with well structured, documented operational source code that can be directly inspected by the Home Health community. This Java source code can be directly compiled without modification, run on any machine supporting Java, and can be incorporated directly into other applications. Another change is that reference data, such as diagnostic category, diagnosis codes, etc. are not embedded into the Java source, but are separate examinable documents that are included during program execution.

Finally, because Java is an Object Oriented language, the design and implementation of the HH-PPS can better reflect the real world understanding, reducing the traditional gap between software requirements and implementation. Java classes or interfaces pertaining to the HH-PPS that are mentioned in this document are detailed in the “javadocs” folder within the installation package. Please refer to the installation package for the exact location of that folder.

Java language

There are many advantages to developing in Java as opposed to C-language. As mentioned above, an advantage of Java is that it is an Object Oriented language that allows developers to design applications that, when implemented, better reflect the real world that the software is intended to represent. For example, if in the real world there is something called a Diagnosis Code and that Code has certain properties (such as value and Diagnostic Category), the software can directly represent that Code as a discrete component that both the customer and the software developer can discuss more effectively. Another advantage is that Java is mature enough to provide many pre-built tools that help standardize program structure and documentation. Probably the most popular advantage is that once Java is written, it can be run on any operating system without change. This greatly reduces the long term maintenance of the application.

Even with these advantages in mind, there is still the fact that software development is part science and part art form. It is true that Object Oriented concepts can be applied to a language like 'C', but it is very difficult and takes plenty of discipline on the part of the developer. By using an Object Oriented language, many of these difficulties are reduced. However, the “art form” is still present.

Designing the real world

The design goal of the Home Health Prospective Payment System (HH-PPS) is to reflect the real world. This version contains several discrete objects (or components) that reflect real world items. Each object is part of a segment of the system. During analysis of the Grouper 2.03 Pseudo code, the following segments of the system emerged:

Data input. Input record required to effectively score.

Data output. Scoring results.

Validation. Ensuring the input data is correct enough to properly score.

Mapping. Only used for V3110 thru V3514**.** Converts data specific to Oasis-C to its Oasis-B equivalence in order to allow the scoring logic to remain equivalent with version v2409. Historical Oasis-B related records do not undergo any mapping.

Scoring. Provides the score based on the validated input data.

References. Tables of data that describe the ICD-9-Diagnosis Codes and their attributes.

Reporting. Provides a means to inspect the grouper's scoring logic.

Management. Provides a means to control the internal operation of the Grouping software.

The following diagram shows the design of the entire system. The design of these segments is described in more detail in the section that follow in this document.

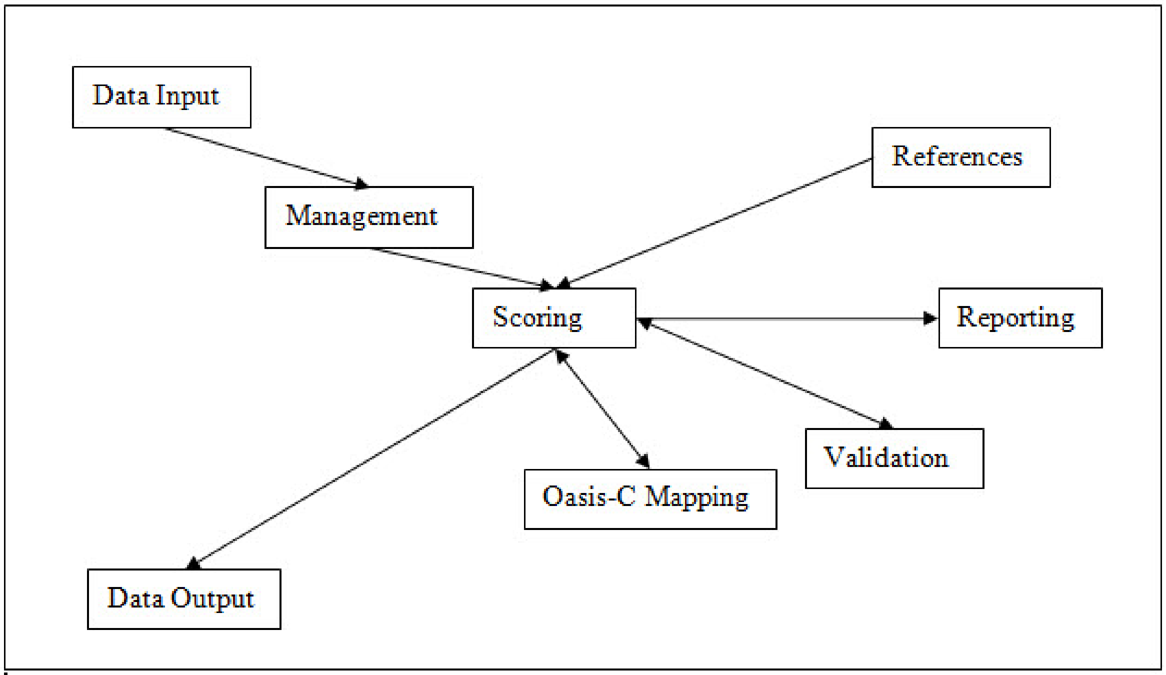

Figure 1: Diagram of overall system design

The overall design of this version is based on the concept of prototyping (Java Interfaces used to define how objects interact within the overall system but without providing the implementation details.) Therefore, by defining the prototype, the functionality of the system is described, but the implementation is left to specific modules. By separating the design from the implementation, flexibility of implementation is increased. This flexibility comes into play as we move forward with new versions. For example, there is an object within this system for representing a Grouper that provides scoring for an input record, version name, etc. For each version, another Grouper implementation will contain the logic specific for that version. The implementations will be used the same way within the system, but each will have its own separate scoring characteristics. Further, both versions are clearly separated, and can be tested and updated without affecting the other.

In describing the design below, the methods for the prototype will provide the minimum functionality for the system. However, implementation of the prototype may include additional methods.

Note: Java interfaces (and classes) are organized into what are referred to as packages. The prototype interfaces below are within the prototype package named:

com.mmm.cms.homehealth.proto

The overall design also uses the following design patterns:

- Factory
- Observer
- Strategy
- Notifier

Data input

This is the data coming into the system and used for scoring. Historically, the input data was a string structured in the OASIS (Outcome and Assessment Information Set) record format (OASIS-B and OASIS-C record formats containing 1448 characters). However, much of the information within the OASIS record is unused in the scoring process. Therefore, this grouper version introduces the definition of a new record subset, the Home Health Record, to hold only the information required for scoring. The Home Health Record can be extended to include all the OASIS record information, but in terms of scoring it would not be necessary. Further, by separating the OASIS record from the scoring record, this allows the scoring record to evolve/change without effecting or relying on the OASIS record format. To support the conversion from the OASIS record to a Home Health Record, the class com.mmm.cms.homehealth.io.OasisBody_B_RecordUtil and com.mmm.cms.homehealth.io.OasisBody_C_RecordUtil provide general conversion methods. To make using these classes easier, an additional class, com.mmm.cms.homehealth.io.OasisReaderFactory, provides the convenient mechanism to determine which …RecordUtil class to use (referred to as the record converter class).

The following diagram shows what the process would be if using the OASIS record as a starting point.

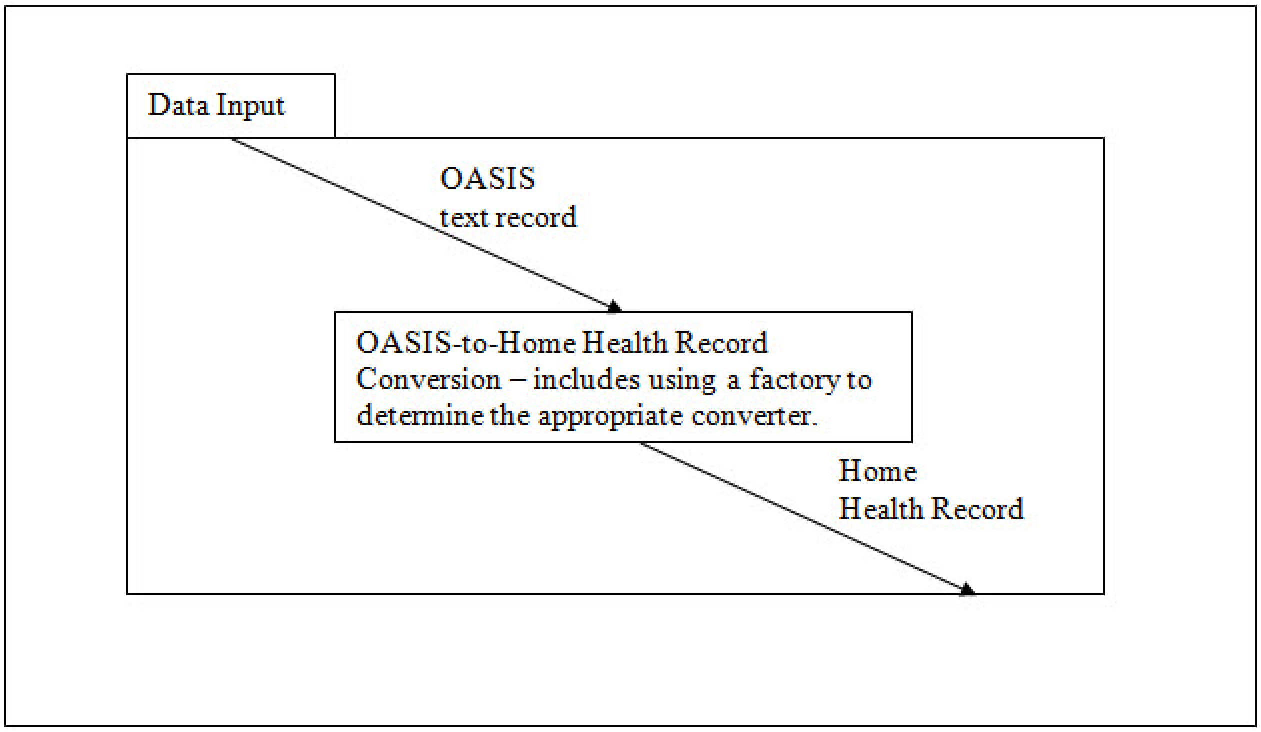
Data Input
OASIS text record
OASIS-to-Home Health Record Conversion – includes using a factory to determine the appropriate converter.
Home Health Record

Figure 2: OASIS to Home Health record conversion process

Interfaces

- **HomeHealthRecord_IF.** This interface defines the information for validating and scoring within the Home Health Prospective Payment System (HH-PPS). This definition requires less information than what would normally be found in a full OASIS-B record.
- **HomeHealthRecord_C_IF.** New in v3110. This interface builds on the HomeHealthRecordIF and defines additional information for validating and scoring within the HH-PPS. This additional information is specific to the OASIS-C validation.
- **OasisRecordConverterIF.** New in v3110. This interface is in the ...io package because it is considered external to the HH-PPS requirements. Its purpose is to provide a generic way of converting an Oasis record string into a HomeHealthRecord_IF implemented object.

Classes

- **OasisReaderFactory.** New in v3110. This class is in the ...io package because it is considered external to the HH-PPS requirements. Its purpose is to provide a mechanism that given an OASIS record string, either a B or C version, determines the appropriate conversion module (i.e. OasisRecordConverterIF) to use in order to read the record in or write it out.

Data output

This is the data that comes out of the system, i.e. the resulting scoring data. The scoring model contains four parts as shown in the following diagram:

- HIPPS (Health Insurance Prospective Payment System) code
- Grouper Version
- Claim OASIS
- Data Validity Flag.

Each of these parts can be treated as discrete objects, but can also be collected to represent the resulting score from the grouper.

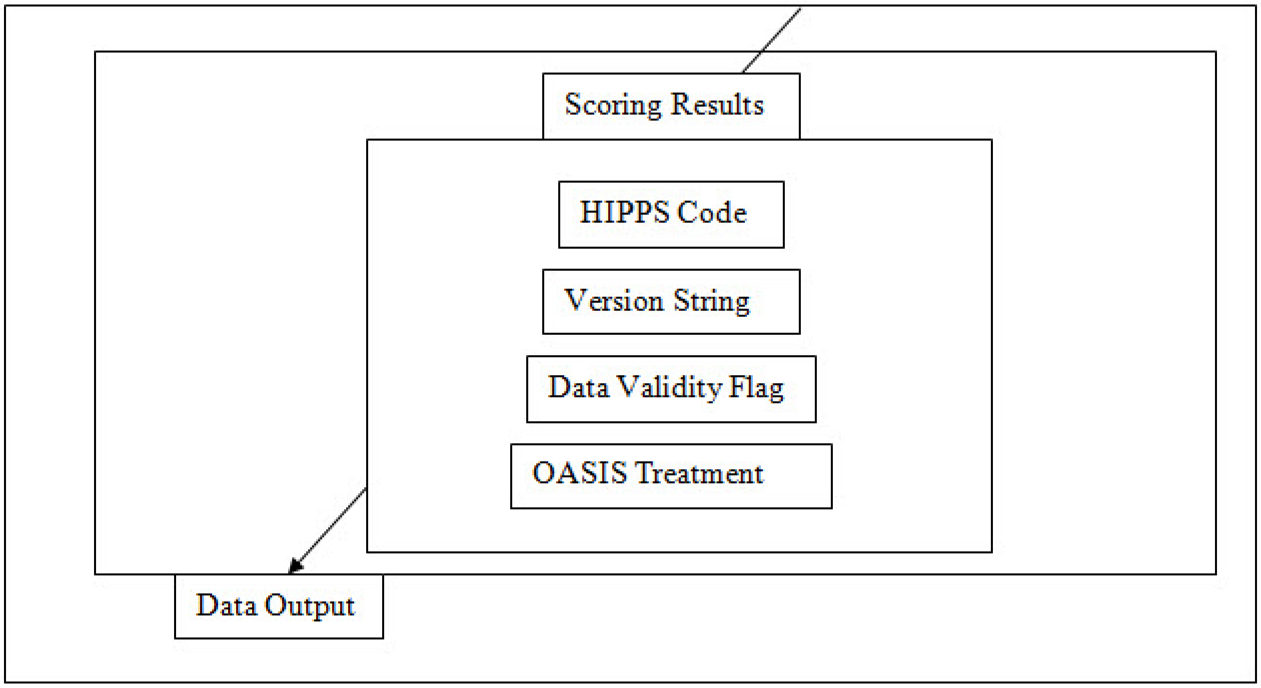

Figure 3: Scoring results and its parts

Interfaces

- **DataValidityFlagIF.** Holds the data issues associated with a processed record, and generates the Validity flag to use in the grouper output.
- **HIPPSCodeIF.** This represents the 5 character code used to report the HIPPS code for the Grouper results.
- **TreatmentAuthorizationIF.** This represents the 18 character code used to report the Oasis Treatment and Authorization for the Grouper results.

Note: the version information comes from the HomeHealthGrouperIF which is not considered an output specific item.

Validation

The interface in this section reviews the Home Health record to determine which values are correct, either individually or as a set of variables. This interface is basically a read-only interface that provides reporting of the validity of the record as well as providing the overall validity flag.

Interfaces

- **HomeHealthRecordValidatorIF.** Provides validation on the Oasis Record and can be performed specific to the version. Once validated, this will indicate what items within the record have issues or are invalid.

Scoring

This is the heart of the grouping system with its main method being score(). This interface takes a new approach by allowing the object to determine for itself if a record is considered eligible for scoring. If the isValidForVersion() returns false then scoring the record will be invalid, and the results are not guaranteed to be accurate.

Interfaces

- **HomeHealthGrouperIF extends Namable, Describable, Initializable, HomeHealthEventNotifierIF.** Describes the high level class that will score the Home Health Record. It manages all its own information about valid Diagnosis, Categories, V-codes, etc. It should organize the information and the Validation and scoring models to perform the detailed scoring process.
- **HomeHealthScoringModelIF.** This represents a single model for scoring a HomeHealth record. Implementations of this class should not have to worry about validating the record, and only focus on scoring it. A Reference to the grouper that created this model at run-time to ensure any reporting through the Notify Events methods can use the grouper instead of implementing an internal version of the notification process. In general, a scoring model has the following phases for processing:

    Initialize (once per object created)

    Score a record

        Pre process the record

        Perform initial calculation

        Re-evaluation the principal codes

        Resolve Etiology / Manifestation Contention for points

        Recalculate point for non Primary Codes

- **ScoringPointsIF.** Holds points accumulated by the Scoring Model. It is up to the model to determine how many scores should be held within.
- **ScoringResultsIF.** Holds the HIPPS (Health Insurance Prospective Payment System) , OASIS (Outcome and Assessment Information Set) Treatment Authorization, Version and Flag information accumulated during the grouper scoring process.

References

References are parts of the application that are not directly involved in scoring but provide information to the score modules, allowing them to perform effectively. For example, the diagnosis codes and their associated Diagnosis Group would be a reference. Case Mix Adjustment tables would also be a reference. The classes involved with these references are designed in such a way that the data itself can actually be external to the application, allowing for easy replacement. The design also does not restrict the location of the external data, however, the implementation may impose such restrictions for deployment.

Interfaces

- **CaseMixAdjustmentItemIF extends Identifiable, Namable, PointsScoringEquationsIF.** This represents a single Casemix Adjustment set. It corresponds to a single row in the original "Table 5 Casemix Adjustment Variables" table of the Pseudo 2.03 Tables spreadsheet.
- **Icd9CodeIF extends Describable.** Represents the basic information about an ICD-9-Code including the code value, its category, and data not required for scoring but useful for GUIs, such as the description.
- **Icd9DiagnosisCodeIF extends Icd9CodeIF, Clonable.** Holds the information specific to the diagnosis codes (as opposed to a procedure code) that supports the scoring.
- **DiagnosticCategoryIF extends Identifiable, Describable.** Holds the ID and the description for a Diagnosis Group.
- **PointsScoringEquationsIF.** This holds a single set of scoring values based on the scoring equations for the combinations of:

    Episode Timing (M0110) - UK or 01 as Early, and 02 as Later

    Therapy Visits (M2200) - two main groups of 0-13, and 14+

These combine to create four separate values, or equations for scoring the case mix. The "equations" are also referred to as a number in the following combination of Episode Timing and Therapy Visits:

Early Time:

0-13 Visits = 1

14+ Visits = 2

Later Time:

0-13 Visits = 3

14+ Visits = 4

The values of each equation can be referred to within either by their name or by their equation number. Note that the NRS (Non-routine medical supplies) scoring will only have one equation available.

Reporting

Reporting is used mainly during the debugging in order to ensure the scoring logic is correct or for capturing the information to present in a user interface. Traditionally, this kind of reporting is detrimental to the performance of the application during deployment. However, object oriented design allows the use of a three-part reporting mechanism that does not adversely affect performance. The three parts are an event notifier, and event listener and the event itself. The HomeHealthGrouperIF and the HomeHealthRecordValidatorIF send the events to any object that may be listening, such as the utility class, the HHEventConsole.

Interfaces

- **HomeHealthEventIF.** Describes the Home Health Event used for notify programmatic listeners during the scoring process.
- **HomeHealthEventListenerIF.** This defines a class the listens for events for the Home Health grouper. These events are used instead of logging.
- **HomeHealthEventNotifierIF.** Defines a Notifier for Home Health Scoring events.

Management

This section is for organizing the grouper versions in a multi-version scoring system. Each record type and assessment can only be scored by a single grouper version. By using this factory, the system can determine which grouper is valid for a record by cycling through its list of available groupers and asking each if it can process the record. The first grouper that replies yes to that question will be the one the factory will select to process the record. While not explicitly defined here, most implementations will allow the grouper version object to be loaded dynamically, which separates this class from being re-programmed when new grouper versions are added to the system.

Interfaces

- **HomeHealthGrouperFactoryIF extends Initializable.** This defines a means for selecting a grouper based on the date range of the Home Health record.

General purpose

General purpose does not target a specific piece of the system. The interfaces defined here are used throughout the system in order to provide a consistent way to treat objects, which is helpful when debugging. The following interfaces are in the package named com.mmm.cms.util

Interfaces

- **Describable.** Defines an object having a getDescription() and setDescription() method.
- **Identifiable.** Defines an object having a getId() and setId().
- **Initializable.** Defines an object having an init() method.
- **Namable.** Defines an object having a getName() and setName().