**Data Collection Types in Java, R, and Python**

Laurence T. Burden

Purdue Global University

IN300: Data Governance

John Brooks

May 16, 2021

Storing information is vital to do some processing on the same information using a programming language. Storing a grouping of information allows for easier processing of data by using a more structured architecture. Many data collection types will allow for looping through the data using a cleaner loop structure than the standard for loop. This paper will look at data collection types in three languages, Java, R, and Python.

**Java Data Collection Types**

All Java collections are built on the Collection interface, which extends the Iterable interface (Collections Framework Overview, 2021). The Iterable interface is what allows the use of for-each loops on Java collections. Three of the available collections are HashSet, ArrayList, and HashMap.

The HashSet collection in Java builds off the Set interface. This collection can store numerous values of the provided data type without duplicates. Some of the standard methods are .add(e) to add an element, .clear() to remove all elements, .contains(Object o) returns whether the set holds the specified object in its collection, and .remove(Object o) to delete an item from the set (HashSet (Java Platform SE 8), 2021 ). An example of creating a set is

**import** java.util.\*;

**public** **class** Expirement {

**public** **static** **void** main(String[] args) {

// Create the set

Set<String> setExample = **new** HashSet<String>();

// Add elements to the set

setExample.add("Hello");

setExample.add("There");

setExample.add("General");

setExample.add("General"); // Attempt to add duplicate value

setExample.add(**null**);

setExample.add("Kenobi");

// Will print [null, Hello, There, General, Kenobi]

// Note that second General is ignored

System.***out***.println(setExample);

}

}

The ArrayList is based on the Java List interface. This collection acts as a dynamic array. Normal Java arrays require their length to be set when instantiated. An ArrayList does not have this limitation. The ArrayList still requires that every data type in the list be the same. Items are added, retrieved, and removed using the .add(), .get(), and .remove() methods respectively. Other methods of note are the .addAll() to add another collection to the current one, the .forEach() method to perform a given action to each element in the ArrayList, and .replaceAll() to replace each element with the result of some action performed on the element (ArrayList (Java Platform SE 8 ), 2021). An example of setting up an array list is as follows.

**import** java.util.\*;

**public** **class** Expirement {

**public** **static** **void** main(String[] args) {

// Create an ArrayList

// Could add number in final parenthesis to create with initial length

ArrayList<Integer> alExample = **new** ArrayList<Integer>();

// Populate the array list

alExample.add(2);

alExample.add(4);

alExample.add(4); // Duplicates are allowed in the Array List

alExample.add(6);

// Will print [2, 4, 4, 6]

System.***out***.println(alExample);

}

}

The final Java collection we will look at is the HashMap. The HashMap acts like a dictionary object. It uses key-value pairs to store information, rather than an assigned index. The HashMap uses .put() instead of .add() for creating new entries. An example of a HashMap is the following.

**import** java.util.\*;

**public** **class** Expirement {

**public** **static** **void** main(String[] args) {

// Create a hash map

HashMap<String, Integer> hmExample = **new** HashMap<String, Integer>();

// Populate has map

hmExample.put("Laurence", 35);

hmExample.put("George", 28);

hmExample.put("Sarah", 31);

// Prints {George=28, Sarah=31, Laurence=35}

// Note that original order is not preserved

System.***out***.println(hmExample);

// Prints 31

System.***out***.println(hmExample.get("Sarah"));

}

}

**R Data Collections**

The R programming language was built for data analysis. Because of this, the creators added a few specialty data collection types. Three of these collections are the vector, list, and matrix.

The R vector is a collection data type. A vector is created using the function c(…). The arguments are all values that should be combined within the vector variable. The c() function will accept any number of values and each can be a different type. Any developer should note that each value will be coerced into a single data type, though. The coercion follows a set chain of data types, with the right-most type being used based on values in the vector (RDocumentation, 2021). The chain of types is NULL < raw < logical < integer < double < complex < character < list < expression (RDocumentation, 2021). An example of a vector in R:

# Create a vector of values

testVect <- c("Some String", 5, FALSE, NULL)

# Prints "Some String" "5" "FALSE"

# Notice that every item has been coerced into a string

print(testVect)

# Create a vector of numbers

numVect <- c(2, 4, 42, 88)

# Prints 2 4 42 88

# Note that the numbers were not coerced into

# a string since all values are the same type

print(numVect)

A list in R is similar to a vector but allows multiple data types. The list is created with the list() function. A list can hold disparate data types and can even hold vectors within it (RDocumentation, 2021). An example of a list is as follows.

# Create a list with many data types

testList <- list("Some String", 5, FALSE, NULL, c(2,3,4))

# Prints the following values, each on a new line

# "Some String" 5 FALSE NULL 2,3,4

# Note that data types were kept as entered

print(testList)

The matrix is the final R data collection type we will look at. The matrix creates a two-dimensional array with optional names for the row and columns. The matrix is created with the following syntax, matrix(data, nrow, ncol, byrow, dimnames). Data is the vector of information to be present in the matrix. Nrow and ncol tell the matrix how many rows and columns to create. Byrow is a Boolean to state whether the input vector is arranged by row. The dimname argument takes a list of two vectors to be used as column and row names. The name vectors must have the same number of elements as the row or column count. If no names are given, then a cartesian number system is used as row and column names (*Matrices*, 2021). An example of an R matrix is:

# Create ow and column names

rowNames <- c("Row 1", "Row 2", "Row 3")

colNames <- c("Col 1", "Col 2", "Col 3")

# Create a 3x3 matrix with values 1-9

testMatrix <- matrix(c(1:9), nrow = 3, dimnames = list(rowNames, colNames))

# Prints

# Col 1 Col 2 Col 3

# Row 1 1 4 7

# Row 2 2 5 8

# Row 3 3 6 9

print(testMatrix)

**Data Collection Types in Python**

Python is a weakly typed language and allows for the mixing of data types in some data collections. There are also multiple ways to create some data collections. Three of Python’s data collection types are the list, set, and dictionary.

Python lists are a grouping of arbitrary Python objects (*3. Data model,* 2020). This allows multiple data types and expressions to be stored in a list. An example of a list is:

# Create a list of different types of objects  
someList = [5, "A String", "A String", True]  
  
# Prints 5 "A String" "A String" True  
# Note duplicates are allowed  
print(someList)

Sets in Python are an unordered group of disparate data types (*3. Data model,* 2020). This means that they cannot be referenced by an index. They are useful because they allow the information to be iterated over. An example of a set being iterated upon is as follows:

# Create a set of numbers  
someSet = {5, 10, 15, 20}  
  
# Prints 15, 25, 10, 20  
# Note the numbers are not kept in order  
for i in someSet:  
 print(i + 5)

A dictionary is a group of objects denoted by a key-value pair (*3. Data model,* 2020). This allows for data retrieval using the key. Dictionaries can be created using a bracket notation or by the dict() constructor. An example of a dictionary is:

# Create a dictionary  
someDict = {  
 1: "Seattle",  
 2: "Detroit",  
 3: "Pittsburgh"  
}  
  
# Prints "Detroit"  
print(someDict[2])

**References:**

Collections Framework Overview. (2021). https://docs.oracle.com/javase/8/docs/technotes/guides/collections/overview.html.

HashSet (Java Platform SE 8 ). (2021, March 19). <https://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html>.

ArrayList (Java Platform SE 8 ). (2021, March 19). <https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html>.

*c: Combine Values into a Vector or List*. RDocumentation. (2021, March). <https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/c>.

*list: Lists -- Generic and Dotted Pairs*. RDocumentation. (2021, March). <https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/list>.

*Matrices*. Tutorials Point. (2021). <https://www.tutorialspoint.com/r/r_matrices.htm#:~:text=Advertisements,of%20the%20same%20atomic%20types>.

*3. Data model*. Python 3.9.5 documentation. (2020, October 5). <https://docs.python.org/3/reference/datamodel.html>.