ADVANCED OOP: CONTAINERS

OVERVIEW

OVERVIEW

- **What is a container?**
  - Something that can store multiple items of same type
    - A public library contains a large number of books
  - Some containers provide some organization to the items
    - The card catalog tells us where to look for each book

- **Java container classes store objects from another class**
  - They can store any type of data
  - They have methods for storing and accessing data
  - They have different representations that greatly effect storage and access speed

(c) Prof. John Gauch, Univ. of Arkansas, 2020

OVERVIEW

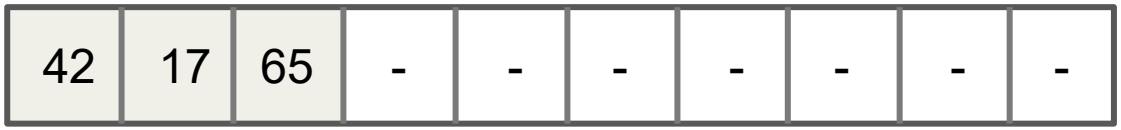
- **Java provides a number of container classes that can store an unlimited number of objects of one data type**
  - Some container classes also provide methods for very rapid data storage and retrieval

- **In this section, we will look into four Java container classes in detail**
  - ArrayList – stores data in a variable length array
  - LinkedList – stores data in a sequence of data nodes
  - HashMap – stores (key, value) pairs in a hash table
  - HashSet – stores data in a hash table without duplicates

(c) Prof. John Gauch, Univ. of Arkansas, 2020

OVERVIEW

- **ArrayList – Stores data in a variable length array where only a portion of the array currently contains data**

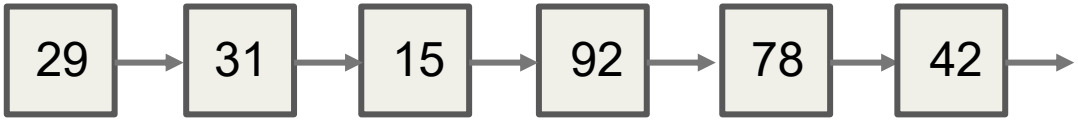


  - As we add data it goes into “unused” section
  - When the “unused” section is full the size of the array automatically grows so we can store more data
  - Data access is as fast and easy like an Array

(c) Prof. John Gauch, Univ. of Arkansas, 2020

OVERVIEW

- **LinkedList – Stores data in a sequence of data storage nodes that are linked together**

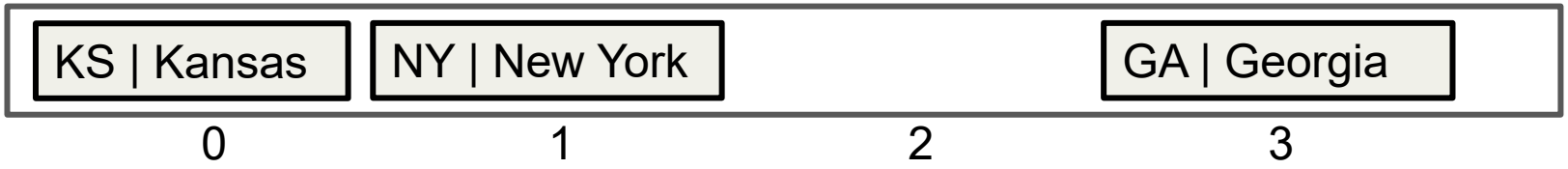


  - As we add data the linked list add nodes to increase the size of this container so we can store the data
  - Data access is much slower than an Array or ArrayList
  - Adding or removing data from linked list can also be slow

(c) Prof. John Gauch, Univ. of Arkansas, 2020

OVERVIEW

- **HashMap – Stores key/value pairs in a hash table and uses the key to quickly access the corresponding value**

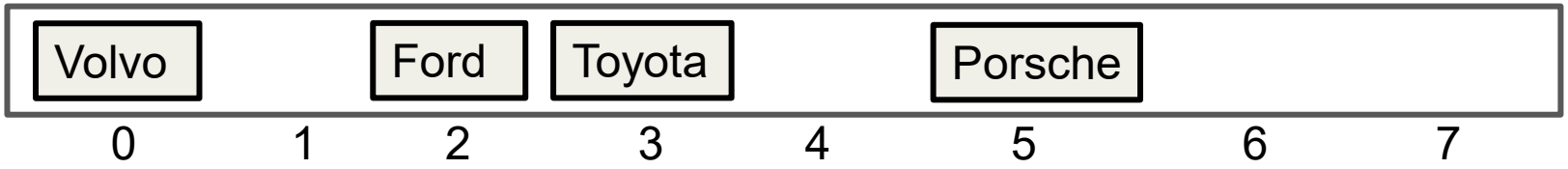


  - A hash function is used to convert the key into the array location for the key/value pair
  - In this example, hash(NY) = 1, so we store (NY, New York) in location 1 of the hash table
  - Data access is even faster than an Array because we can jump immediately to desired value

(c) Prof. John Gauch, Univ. of Arkansas, 2020

OVERVIEW

- **HashSet – Stores data values in a hash table and also provides mathematical Set operations**



  - A hash function is used to convert the data value into the array location for the data value
  - In this example, hash("Porsche") = 5, so we store "Porsche" at location 5 of the hash table
  - Duplicate data values are not stored in a HashSet
  - Set operations are very fast compared to using an Array

(c) Prof. John Gauch, Univ. of Arkansas, 2020

OVERVIEW

- **Lesson objectives:**
  - Learn how ArrayList can be defined and used
  - Learn how LinkedList can be defined and used
  - Learn how HashMap can be defined and used
  - Learn how HashSet can be defined and used
  - Study sample programs using these container classes

(c) Prof. John Gauch, Univ. of Arkansas, 2020

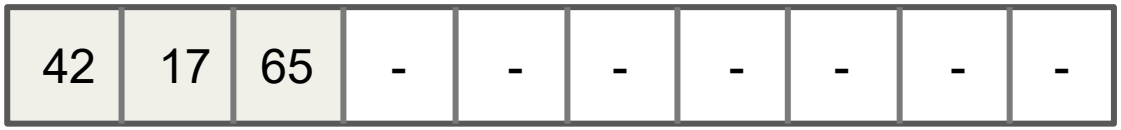
ADVANCED OOP: CONTAINERS

PART 1

ARRAY LIST

ARRAY LIST

- **The ArrayList class is a resizable Array that is defined in the java.util package**



  - A built-in array is fixed size
  - When the array becomes full, you have to manually create another array and copy the data
  - An ArrayList is not fixed size
  - When the ArrayList becomes full, the container will automatically resize itself to be 50% larger

(c) Prof. John Gauch, Univ. of Arkansas, 2020

ARRAY LIST

- **How to declare an ArrayList**

```
import java.util.ArrayList;
public class Main
{
   public static void main(String[] args)
   {
      ArrayList<String> cars = new ArrayList<String>();
      ...
   }
}
```

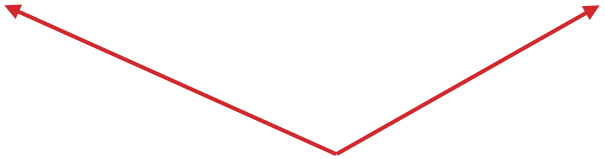

Bring in the ArrayList class

(c) Prof. John Gauch, Univ. of Arkansas, 2020

ARRAY LIST

- **How to declare an ArrayList**

```
import java.util.ArrayList;
public class Main
{
   public static void main(String[] args)
   {
      ArrayList<String> cars = new ArrayList<String>();
      ...
   }
}
```



Declare an ArrayList
to store Strings

(c) Prof. John Gauch, Univ. of Arkansas, 2020

ARRAY LIST

- **How to declare an ArrayList**

```
import java.util.ArrayList;
public class Main
{
   public static void main(String[] args)
   {
      ArrayList<String> cars = new ArrayList<String>();
      ...
   }
}
```

We do not need to give a size for the ArrayList

ARRAY LIST

- **How to store data in an ArrayList**

```
ArrayList<String> cars = new ArrayList<String>();
cars.add("Volvo");
cars.add("BMW");
cars.add("Ford");
cars.add("Mazda");
```

The array ArrayList
now contains four cars
in locations 0,1,2,3

(c) Prof. John Gauch, Univ. of Arkansas, 2020

ARRAY LIST

- **How to access and print data in an ArrayList**

```
ArrayList<String> cars = new ArrayList<String>();
cars.add("Volvo");
cars.add("BMW");
cars.add("Ford");
cars.add("Mazda");
for (int i = 0; i < cars.size(); i++)
   System.out.println(cars.get(i));
```

The size method tells us how many values are stored in the ArrayList

(c) Prof. John Gauch, Univ. of Arkansas, 2020

ARRAY LIST

- **How to access and print data in an ArrayList**

```
ArrayList<String> cars = new ArrayList<String>();
cars.add("Volvo");
cars.add("BMW");
cars.add("Ford");
cars.add("Mazda");
for (int i = 0; i < cars.size(); i++)
   System.out.println(cars.get(i));
```

The get method lets us access locations 0,1,2,3

(c) Prof. John Gauch, Univ. of Arkansas, 2020

ARRAY LIST

- **How to access and print data in an ArrayList**

```
ArrayList<String> cars = new ArrayList<String>();
cars.add("Volvo");
cars.add("BMW");
cars.add("Ford");
cars.add("Mazda");
System.out.println(cars);
```

This will print [Volvo, BMW, Ford, Mazda]

(c) Prof. John Gauch, Univ. of Arkansas, 2020

ARRAY LIST

- **How to remove one data value from an ArrayList**

```
ArrayList<String> cars = new ArrayList<String>();
cars.add("Volvo");
cars.add("BMW");
cars.add("Ford");
cars.add("Mazda");
cars.remove(1);
System.out.println(cars);
```

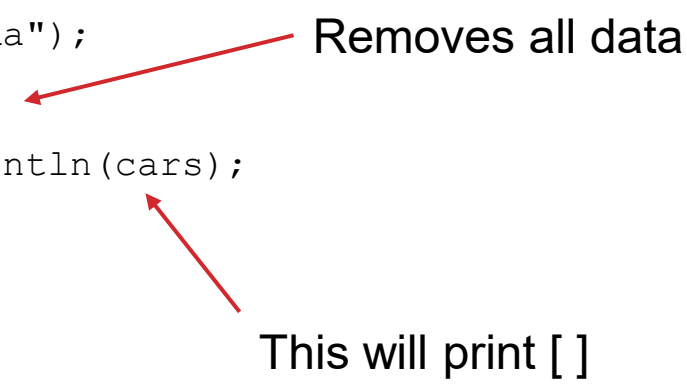
This will remove data from location 1

This will print [Volvo, Ford, Mazda]

ARRAY LIST

- **How to remove all data from an ArrayList**

```
import java.util.Collections;
...
ArrayList<String> cars = new ArrayList<String>();
cars.add("Volvo");
cars.add("BMW");
cars.add("Ford");
cars.add("Mazda");
cars.clear();
System.out.println(cars);
```



(c) Prof. John Gauch, Univ. of Arkansas, 2020

ARRAY LIST

- **How to sort data in an ArrayList**

```
import java.util.Collections;
...
ArrayList<String> cars = new ArrayList<String>();
cars.add("Volvo");
cars.add("BMW");
cars.add("Ford");
cars.add("Mazda");
Collections.sort(cars);
System.out.println(cars);
```

Bring in the Collections class

(c) Prof. John Gauch, Univ. of Arkansas, 2020

ARRAY LIST

- **How to sort data in an ArrayList**

```
import java.util.Collections;
...
ArrayList<String> cars = new ArrayList<String>();
cars.add("Volvo");
cars.add("BMW");
cars.add("Ford");
cars.add("Mazda");
Collections.sort(cars);
System.out.println(cars);
```

Sort the data in alphabetical order

This will print [BMW, Ford, Mazda, Volvo]

(c) Prof. John Gauch, Univ. of Arkansas, 2020

ARRAY LIST

- **See the full list of methods in the ArrayList class here:**

https://docs.oracle.com/javase/10/docs/api/java/util/ArrayList.html

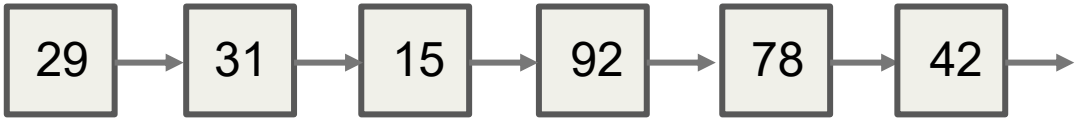
ADVANCED OOP: CONTAINERS

PART 2

LINKED LIST

LINKED LIST

- **The LinkedList class uses a node based representation to store variable amount of data**

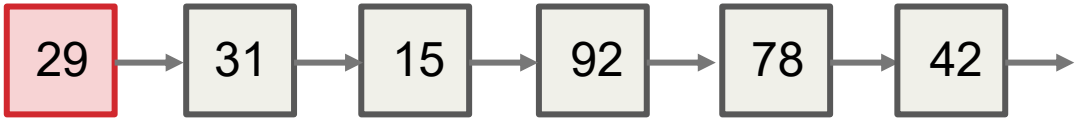


  - Each node contains one data element
  - Each node is “connected” by a reference (shown as arrows) to the next node to the right in the linked list

(c) Prof. John Gauch, Univ. of Arkansas, 2020

LINKED LIST

- **The LinkedList class uses a node based representation to store variable amount of data**

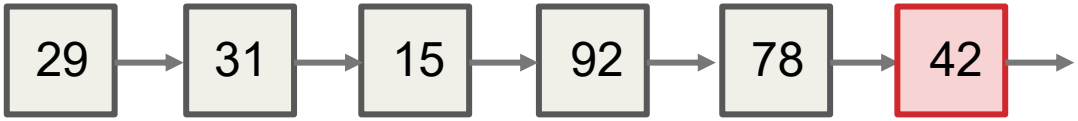


  - Each node contains one data element
  - Each node is "connected" by a reference (shown as arrows) to the next node to the right in the linked list
  - The node on far left is the "head" of the list

(c) Prof. John Gauch, Univ. of Arkansas, 2020

LINKED LIST

- **The LinkedList class uses a node based representation to store variable amount of data**

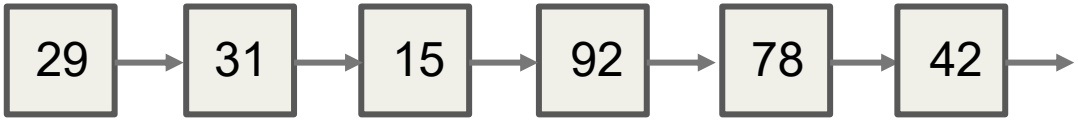


  - Each node contains one data element
  - Each node is "connected" by a reference (shown as arrows) to the next node to the right in the linked list
  - The node on far left is the "head" of the list
  - The node on far right is the "tail" of the list

(c) Prof. John Gauch, Univ. of Arkansas, 2020

LINKED LIST

- **How do we use a LinkedList to store data?**



  - To access data, we must start at head and “walk the list” from node to node to desired node
  - To add data, we create a new node and connect it at the desired location (often before head or after tail)
  - To remove data, we must find the desired node, delete it, and connect the node before to node after

(c) Prof. John Gauch, Univ. of Arkansas, 2020

LINKED LIST

- **Cons of a LinkedList?**
  - It does not support random access like an Array or ArrayList so data access can be much slower
  - It is slow and complicated to insert/delete new data in the middle of the LinkedList

- **Pros of a LinkedList?**
  - It can grow/shrink to meet exact storage needs
  - Fast and easy to insert/delete data at head or tail
  - We can insert data anywhere without having to move data
  - It has all of the methods in the ArrayList class

(c) Prof. John Gauch, Univ. of Arkansas, 2020

LINKED LIST

- **How to declare a LinkedList**

```
import java.util.LinkedList;
public class Main
{
   public static void main(String[] args)
   {
      LinkedList<String> cars = new LinkedList<String>();
      ...
   }
}
```

Bring in the LinkedList class

Declare a LinkedList
to store Strings

(c) Prof. John Gauch, Univ. of Arkansas, 2020

LINKED LIST

- **How to add and remove data in a LinkedList**

```
LinkedList<String> cars = new LinkedList<String>();
cars.add("Volvo");
cars.add("BMW");
cars.add("Ford");
cars.add("Mazda");
cars.remove(1);
System.out.println(cars);
```

This will add four data values

This will remove data from location 1

This will print [Volvo, Ford, Mazda]

(c) Prof. John Gauch, Univ. of Arkansas, 2020

LINKED LIST

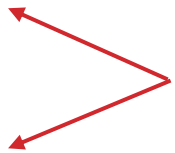
- **The LinkedList class has very fast methods for adding and removing at the head or tail of the linked list**
  - addFirst() - Adds an item before head of the list.
  - addLast() - Adds an item after tail of the list
  - removeFirst() - Remove item from the head of the list.
  - removeLast() - Remove item from the tail of the list
  - getFirst() - Get the item at the head of the list
  - getLast() - Get the item at the tail of the list

(c) Prof. John Gauch, Univ. of Arkansas, 2020

LINKED LIST

- **How to add at beginning and end of a LinkedList**

```
LinkedList<String> cars = new LinkedList<String>();
cars.addFirst("Volvo");
cars.addLast("BMW");
cars.addFirst("Ford");
cars.addLast("Mazda");
System.out.println(cars);
```

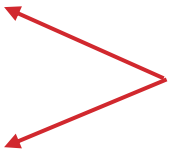


This adds to the beginning of linked list

LINKED LIST

- **How to add at beginning and end of a LinkedList**

```
LinkedList<String> cars = new LinkedList<String>();
cars.addFirst("Volvo");
cars.addLast("BMW");
cars.addFirst("Ford");
cars.addLast("Mazda");
System.out.println(cars);
```



This adds to the end of linked list

LINKED LIST

- **How to add at beginning and end of a LinkedList**

```
LinkedList<String> cars = new LinkedList<String>();
cars.addFirst("Volvo");
cars.addLast("BMW");
cars.addFirst("Ford");
cars.addLast("Mazda");
System.out.println(cars);
```

This will print [Ford, Volvo, BMW, Mazda]

(c) Prof. John Gauch, Univ. of Arkansas, 2020

LINKED LIST

- **See the full list of methods in the LinkedList class here:**

https://docs.oracle.com/javase/10/docs/api/java/util/LinkedList.html

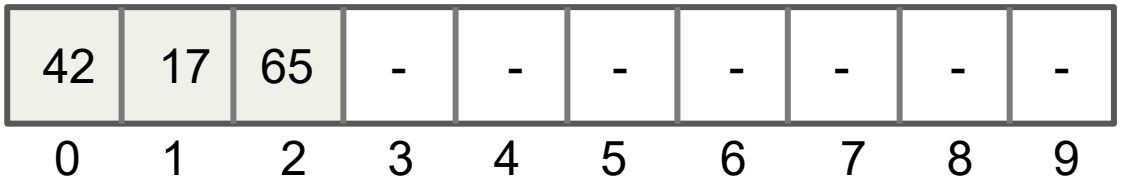
ADVANCED OOP: CONTAINERS

PART 3

HASH MAP

HASH MAP

- **The data elements in an Array or ArrayList can be accessed with an integer index number**

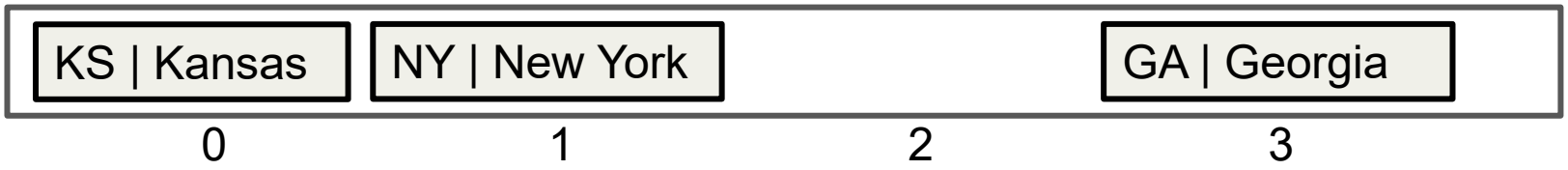


  - Once we know the index, we can access the data value
  - In this case, Data[0] = 42, Data[1] = 17, Data[2] = 65
  - Unfortunately, with an Array there is no way to know in advance where a given data value will be stored
  - If we want to find the 17 we have to loop over all possible index numbers and check the Array at that location

(c) Prof. John Gauch, Univ. of Arkansas, 2020

HASH MAP

- **The data elements in a HashMap are stored in key/value pairs and can be directly accessed using the key**



  - A "hash function" is used to convert keys into locations
  - hash("KS") = 0, so (KS, Kansas) is stored in location 0
  - hash("NY") = 1, so (NY, New York) is stored in location 1
  - hash("GA") = 3, so (GA, Georgia) is stored in location 3
  - Once we know hash(key), we can jump immediately to desired key/value pair in the HashMap

(c) Prof. John Gauch, Univ. of Arkansas, 2020

HASH MAP

- **The keys for HashMap must be Java objects**
  - We can not use primitive types like int, double, char
  - Instead, we must use corresponding “wrapper classes” Integer, Double, Character

- **Java will allow you to use any combination of data types for the value and the key**
  - String value, Integer key
  - Double value, String key
  - Integer value, Student key

HASH MAP

- **Declaring a HashMap:**

```
import java.util.HashMap;
public class Main
{
   public static void main(String[] args)
   {
      HashMap<String, String> capitals = new HashMap<String, String>();
      ...
   }
}
```

Bring in the HashMap class

Declare a HashMap to store String keys and String values

(c) Prof. John Gauch, Univ. of Arkansas, 2020

HASH MAP

- **Adding items to a HashMap:**

```
public static void main(String[] args)
{
    HashMap<String, String> capitals = new HashMap<String, String>();
    capitals.put("England", "London");
    capitals.put("Germany", "Berlin");
    capitals.put("Norway", "Oslo");
    capitals.put("USA", "Washington DC");
    ...
```

This will store (key, value) pairs in the HashMap

HASH MAP

- **Printing items in a HashMap:**

```
public static void main(String[] args)
{
    HashMap<String, String> capitals = new HashMap<String, String>();
    capitals.put("England", "London");
    capitals.put("Germany", "Berlin");
    capitals.put("Norway", "Oslo");
    capitals.put("USA", "Washington DC");
    System.out.println(capitals);
    ...
```

This will print all of the (key, value) pairs {USA=Washington DC, Norway=Oslo, England=London, Germany=Berlin}

(c) Prof. John Gauch, Univ. of Arkansas, 2020

HASH MAP

- **Accessing items in a HashMap:**

```
public static void main(String[] args)
{
    HashMap<String, String> capitals = new HashMap<String, String>();
    capitals.put("England", "London");
    capitals.put("Germany", "Berlin");
    capitals.put("Norway", "Oslo");
    capitals.put("USA", "Washington DC");
    System.out.println(capitals.get("England"));
    ...
```

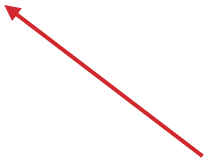
This will get the value associated with the key “England” and print “London”

(c) Prof. John Gauch, Univ. of Arkansas, 2020

HASH MAP

- **Removing items from a HashMap:**

```
public static void main(String[] args)
{
    HashMap<String, String> capitals = new HashMap<String, String>();
    capitals.put("England", "London");
    capitals.put("Germany", "Berlin");
    capitals.put("Norway", "Oslo");
    capitals.put("USA", "Washington DC");
    capitals.remove("USA");
    ...
```



This will remove the (key, value) pair that has “USA” as the key

(c) Prof. John Gauch, Univ. of Arkansas, 2020

HASH MAP

- **Looping over items in a HashMap:**

```
public static void main(String[] args)
{
    HashMap<String, String> capitals = new HashMap<String, String>();
    ...
    for (String i : capitals.keySet())
    {
        System.out.println(i);
    }
}
```

This will return a Set with all HashMap keys that we can iterate over and print

HASH MAP

- **Looping over items in a HashMap:**

```
public static void main(String[] args)
{
    HashMap<String, String> capitals = new HashMap<String, String>();
    ...
    for (String i : capitals.values())
    {
        System.out.println(i);
    }
}
```

This will return a Collection with all HashMap values that we can iterate over and print

HASH MAP

- **Looping over items in a HashMap:**

```
public static void main(String[] args)
{
    HashMap<String, String> capitals = new HashMap<String, String>();
    ...
    for (String i : capitals.keySet())
    {
        System.out.println("key: " + i + " value: " + capitals.get(i));
    }
}
```

We can iterate over all keys to access and print all (key, value) pairs in the HashMap

HASH MAP

- **Summary of common HashMap methods:**
  - put(key, value) – adds (key, value) to the HashMap
  - get(key) – returns value for this key from HashMap
  - remove(key) – removes (key, value) from HashMap
  - clear() – removes all (key, value) pairs from HashMap
  - size() – returns the number of (key, value) pairs
  - isEmpty() – returns true if size == 0
  - keySet() – returns a Set containing all keys
  - values() – returns a Collection containing all values

(c) Prof. John Gauch, Univ. of Arkansas, 2020

CODE DEMO

CountWords2.java

CountWords3.java

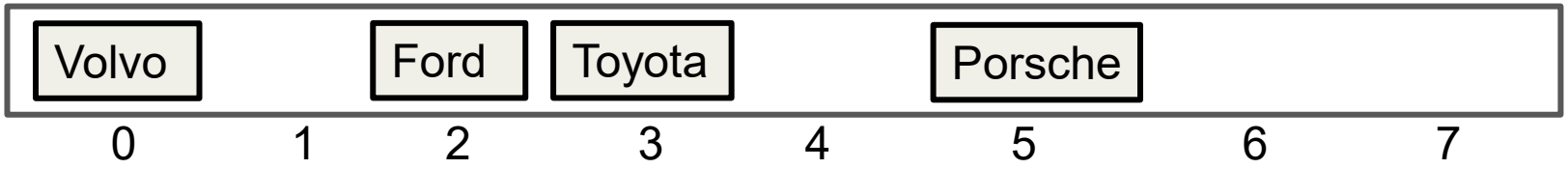
ADVANCED OOP: CONTAINERS

PART 4

HASH SET

HASH SET

- **A HashSet stores data values in a hash table and also provides all mathematical Set operations**



  - Like a HashMap, a “hash function” is used to determine the location for the data value in the hash table
  - hash(“Porsche”) = 5, so “Porsche” is stored in location 5
  - Duplicate data values are not stored in a HashSet
  - Operations are very fast compared to a regular Set

(c) Prof. John Gauch, Univ. of Arkansas, 2020

HASH SET

- **Declaring a HashSet:**

```
import java.util.HashSet;
public class Main
{
   public static void main(String[] args)
   {
      HashSet<String> cars = new HashSet<String>();
      ...
   }
}
```

Bring in the HashSet class

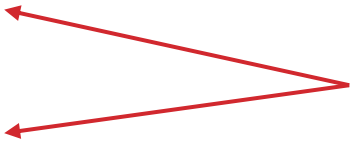
Declare a HashSet to store String values

(c) Prof. John Gauch, Univ. of Arkansas, 2020

HASH SET

- **Adding items to a HashSet:**

```
public static void main(String[] args)
{
    HashSet<String> cars = new HashSet<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("BMW");
    cars.add("Mazda");
    System.out.println(cars);
```



This will store car names in the HashSet and ignore duplicate values

HASH SET

- **Printing items in a HashSet:**

```
public static void main(String[] args)
{
    HashSet<String> cars = new HashSet<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("BMW");
    cars.add("Mazda");
    System.out.println(cars);
```

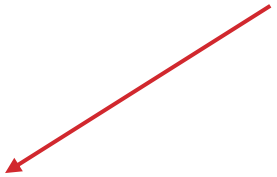
This will print all the car names in the HashSet
[Volvo, Mazda, Ford, BMW]

(c) Prof. John Gauch, Univ. of Arkansas, 2020

HASH SET

- **Checking if an item is in a HashSet:**

```
public static void main(String[] args)
{
    HashSet<String> cars = new HashSet<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("BMW");
    cars.add("Mazda");
    if (cars.contains("BMW")
        System.out.println("Found BMW");
```



This will return true if "BMW" is in the HashSet The hash table makes this operation very fast

HASH SET

- **Removing an item from a HashSet:**

```
public static void main(String[] args)
{
    HashSet<String> cars = new HashSet<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("BMW");
    cars.add("Mazda");
    cars.remove("Ford");
    System.out.println(cars);
```

This will remove “Ford” from the HashSet and print [Volvo, Mazda, BMW]

(c) Prof. John Gauch, Univ. of Arkansas, 2020

HASH SET

- **Looping over items in a HashSet:**

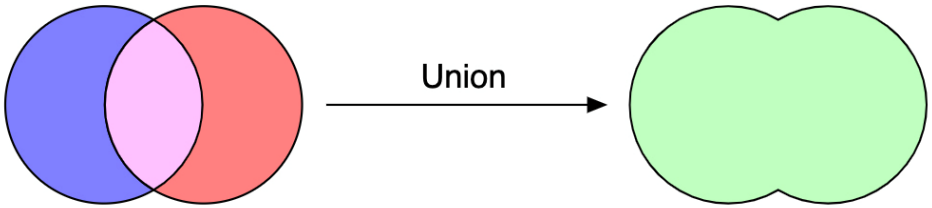
```
public static void main(String[] args)
{
    HashSet<String> cars = new HashSet<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("BMW");
    cars.add("Mazda");
    for (String i : cars)
        System.out.println(i);
```

This will iterate over the Set of cars and print each value on a separate line

(c) Prof. John Gauch, Univ. of Arkansas, 2020

HASH SET

- **Set union combines all elements in in two sets**



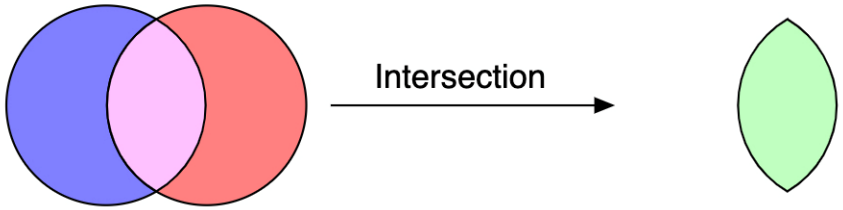

```
HashSet<Integer> A = new HashSet<Integer>();
HashSet<Integer> B = new HashSet<Integer>();
A.add(1); A.add(2); A.add(3);
B.add(2); B.add(4); B.add(6);
A.addAll(B);
System.out.println(A);
```

This will calculate the Set union and print [1, 2, 3, 4, 6]

(c) Prof. John Gauch, Univ. of Arkansas, 2020

HASH SET

- **Set intersection finds common elements in two sets**




```
HashSet<Integer> A = new HashSet<Integer>();
HashSet<Integer> B = new HashSet<Integer>();
A.add(1); A.add(2); A.add(3);
B.add(2); B.add(4); B.add(6);
A.retainAll(B);
System.out.println(A);
```

This will calculate the Set intersection and print [1, 3]

(c) Prof. John Gauch, Univ. of Arkansas, 2020

HASH MAP

- **Summary of common HashSet methods:**
  - add(value) – adds value to the HashSet
  - contains(key) – returns true value is in HashSet
  - remove(key) – removes value from HashSet
  - clear() – removes all values from HashSet
  - size() – returns the number of values in HashSet
  - isEmpty() – returns true if size == 0
  - addAll(Set s) – adds all elements of s to this HashSet
  - removeAll(Set s) – removes all elements of s from HashSet
  - retainAll(Set s) – removes all elements that are not from set s
  - equals(Set s) – returns true if every member of the given set is contained in this set

(c) Prof. John Gauch, Univ. of Arkansas, 2020

SUMMARY

- **The container classes in Java are very powerful because they allow us to store and access large collections of data**
  - They grow and shrink in size to meet the data storage needs of an application
  - They provide very specialized data access methods that can be very fast (hash table) or very slow (linked list)

- **See the Java documentation for more details**
  - https://docs.oracle.com/en/java/javase/16
  - https://docs.oracle.com/javase/tutorial/index.html
  - https://www.w3schools.com/java/default.asp

(c) Prof. John Gauch, Univ. of Arkansas, 2020