Object Oriented Programming with Java

CHAP 1: JAVA PROGRAMMING BASICS

Collections Part One: By Aphrodice Rwagaju

Primitive arrays

- You can create “primitive” arrays in Java using the [ ] syntax
- These are a very convenient way to group some objects together
- But they are limited in functionality
  - You can’t add new items
  - You can’t remove existing items

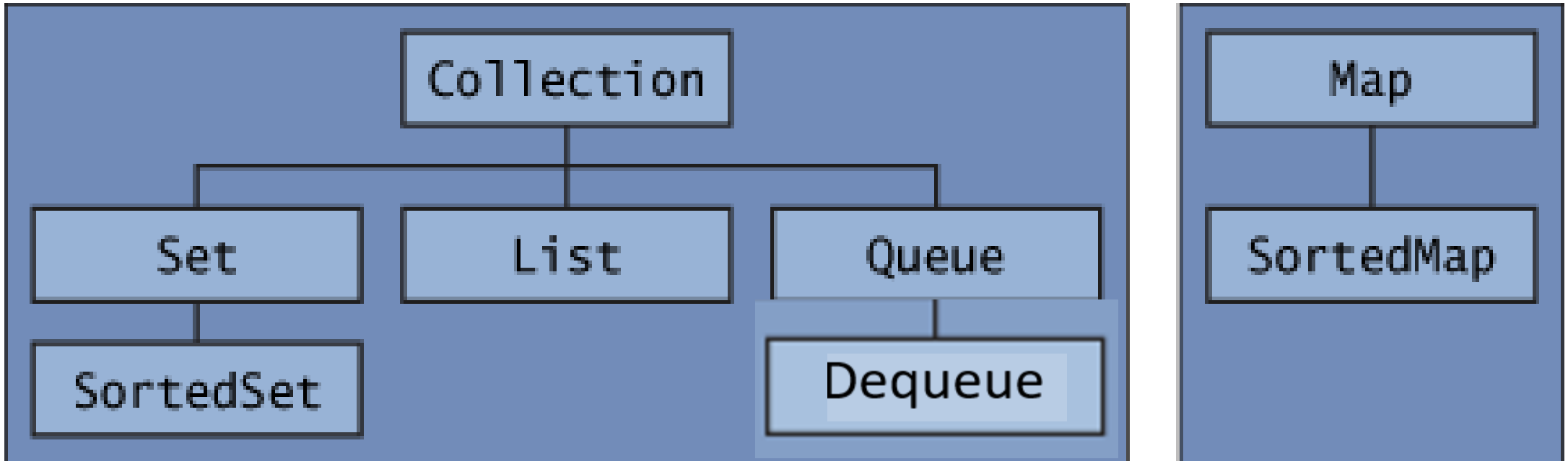
```
int[] vals = new int[4];
vals[0] = 1234;
```

Collections

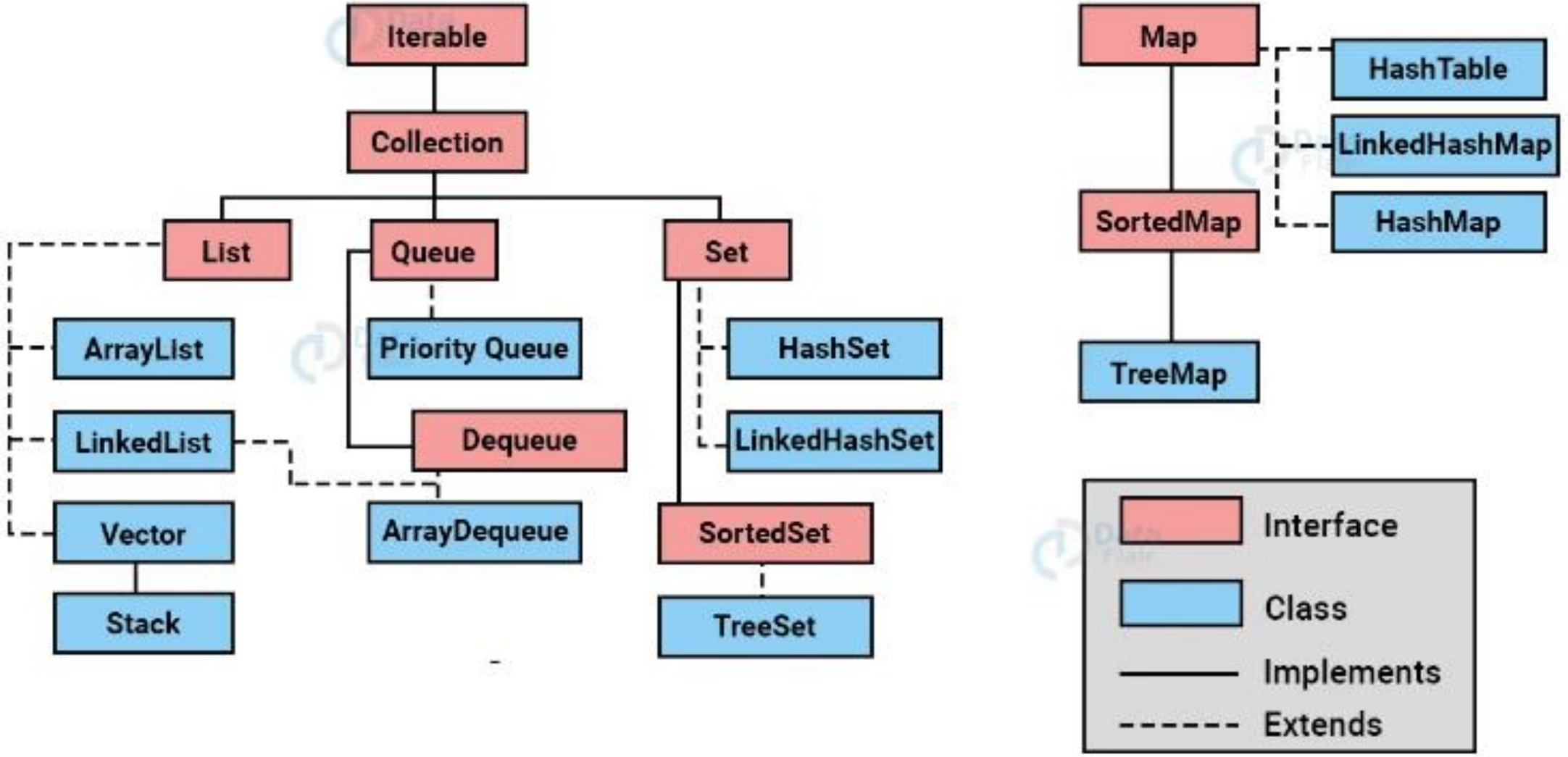
- We can create new classes that provide this extra functionalities, or…
- Java has many different collection classes included so we can use one of them
- Different collection types are good for different things

JDK collections

- The JDK contains a set of **interfaces** for each of the main collection types



JDK collections – Interfaces and Classes



Sets vs lists

- Sets don’t allow duplicates, e.g.
  - {1, 2, 3} can be a set but {1, 2, 1} can’t
- Order is not important
  - {1, 2, 3} is the same set as {3, 2, 1}
- Lists allow duplicates because order is significant in a list
- Items can be inserted at a specific location

ArrayList

- `List` is just an **interface – you can't create an instance of `List`**

- `ArrayList` implements `List` and provides its functionality like an array

```
ArrayList values = new ArrayList();
values.add(1235);
values.add(457);
```

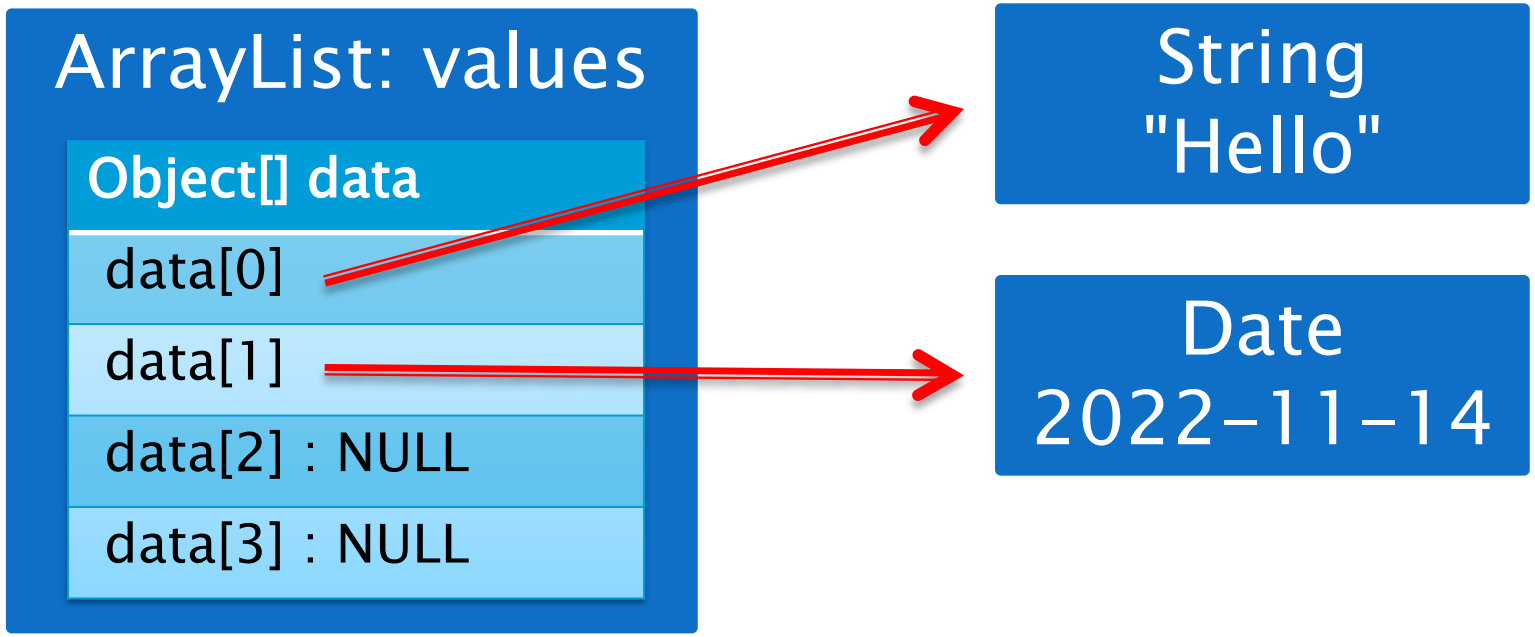
Array dynamically grows

Raw collections

- One day you will learn about **generics** which is the right way to use collections
- For now we use "**raw**" versions of the collection classes
- These store every item as an `Object` reference variable
- You can add any type or object to such a collection

ArrayList example

```
ArrayList values = new ArrayList();
values.add(new String("Hello"));
values.add(new Date());
```



ArrayList example

- Every item is stored as an object reference
- When we fetch items from the array, we get back objects, e.g.

```
ArrayList values = new ArrayList();
values.add(new String("Hello"));
values.add(new Date());

Object obj1 = values.get(0);

String string = (String)values.get(0);
```

Use a cast to get original object

Boxing

- A primitive type (`int`, `bool` etc) cannot be stored using an object reference
- Instead these have to be converted to an equivalent boxed type, e.g.

```
Object o1 = new Integer(123);
Object o2 = new Character('x');
```

- Thankfully Java provides automatic boxing so you can write..

```
Object o1 = 123;
Object o2 = 'x';
```

Becomes an `Integer` object

ArrayList example

- So when you add primitive values to a collection, they get boxed into objects, e.g.

```
ArrayList values = new ArrayList();
values.add(123);
values.add('x');

Integer x = (Integer)values.get(0);

int y = (Integer)values.get(0);

int z = values.get(0);
```

Boxed into an `Integer` object

Unboxed to a `int` value

Error – can't unbox `Object`

HashSet

- This class provides a Set implementation using hash codes to check for duplicates

```
HashSet set = new HashSet();
set.add(1234);
set.add("Hello");
set.add(1234);

System.out.println(set.size());  // Will print "2"
```

- Sets don't have a get(…) method…

HashSet access

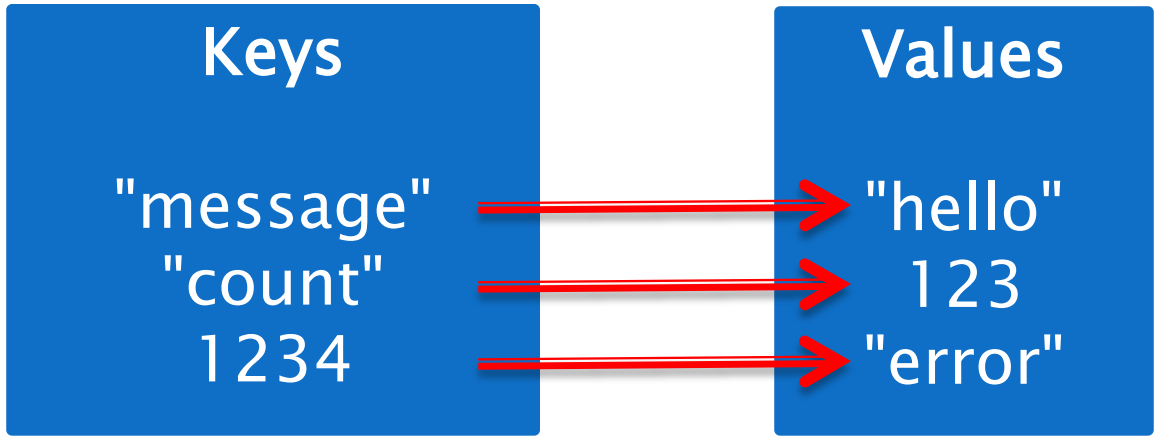
- We can't get an item by its index because items in a set don't have an order
- We can *iterate* through the set to get all the items, e.g.

```
HashSet set = new HashSet();
set.add("Hello");
set.add("World");

for (Object obj : set)
    System.out.println(obj);
```

Maps

- Part of the "Java collections framework" but they don't extend the `Collection` interface
- Similar to `array()` in PHP
- A mapping of keys to values…



Map example

- Maps are useful for storing name/value pairs
- The names are the map keys, e.g.

```
HashMap map = new HashMap();
map.put("message", "Hello");
map.put("count", 123);
map.put("name", "Juru Heaven");

String message = (String)map.get("message");
int count = (Integer)map.get("count");
String name = (String)map.get("name");
```

More about collections…

http://download.oracle.com/javase/tutorial/collections/index.html

https://data-flair.training/blogs/collection-framework-in-java/

EoF