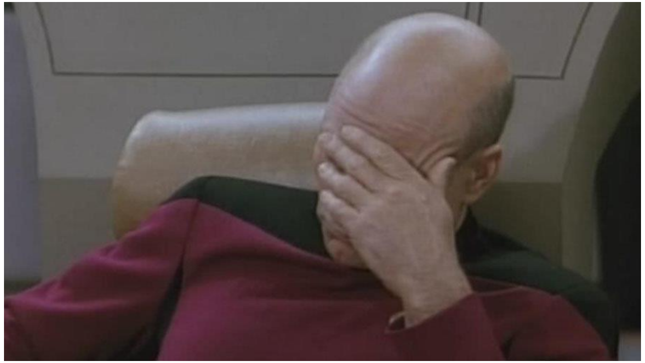
Why did the Java Developer quit his job?

Because he didn't get arrays.



Module 1-7

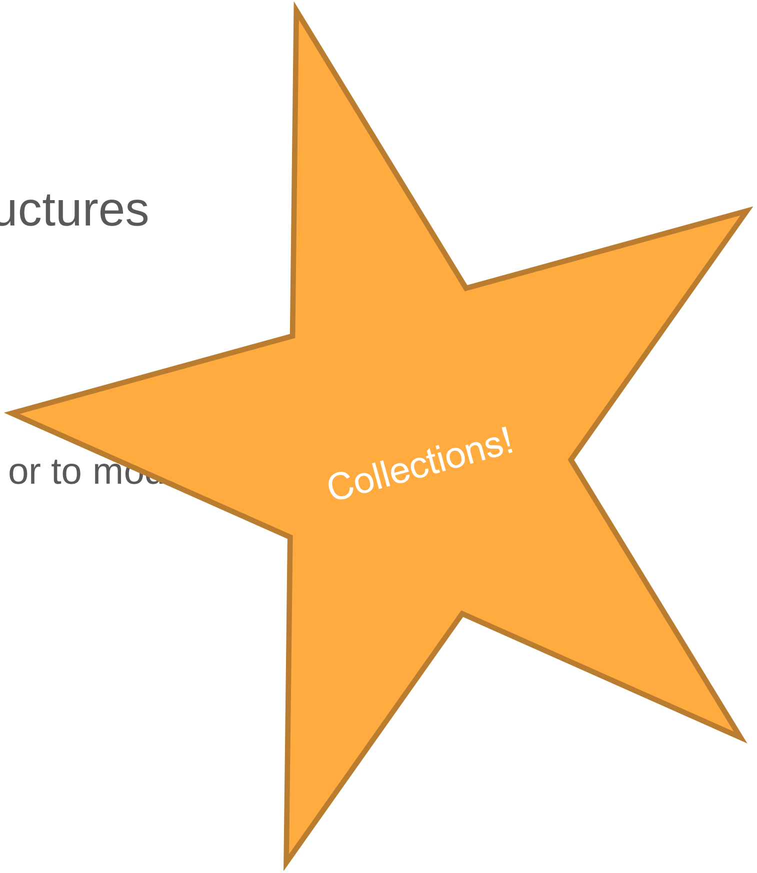
Collections: Lists - ArrayLists

Objectives

1. Differences between array and ArrayList
2. What are Collections and why we use them
3. Packages in Java for organization
4. Stack and Queue

Array Recap

- Arrays are simple data structures
  - Hold collection of like data
- Not flexible
  - Difficult to add new element or to mod



Collections!

Array vs ArrayList

JAVA

Declaration

Array	ArrayList
int arr[] = new int[10]	ArrayList<Type>arrL=new ArrayList<Type>();

Resizable

Array	ArrayList
Static in size	Dynamic in size
Can not change the length after creating the Array object.	As elements are added to an ArrayList its capacity grows automatically.

Primitives

Array	ArrayList
can contain both primitive data types as well as objects.	can not contains primitive data types (like int , float , double) it can only contains Object

Length

Array	ArrayList
.length	.size()

Adding Elements

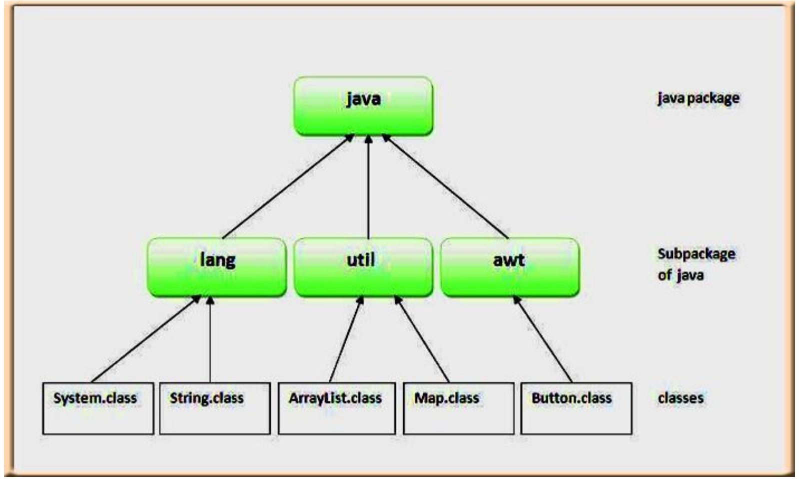
Array	ArrayList
Assignment operator	.add()

Multidimensional

Array	ArrayList
Can be multi dimensional	Always single dimensional.

Package

- Understand use of packages in Java to help organize libraries

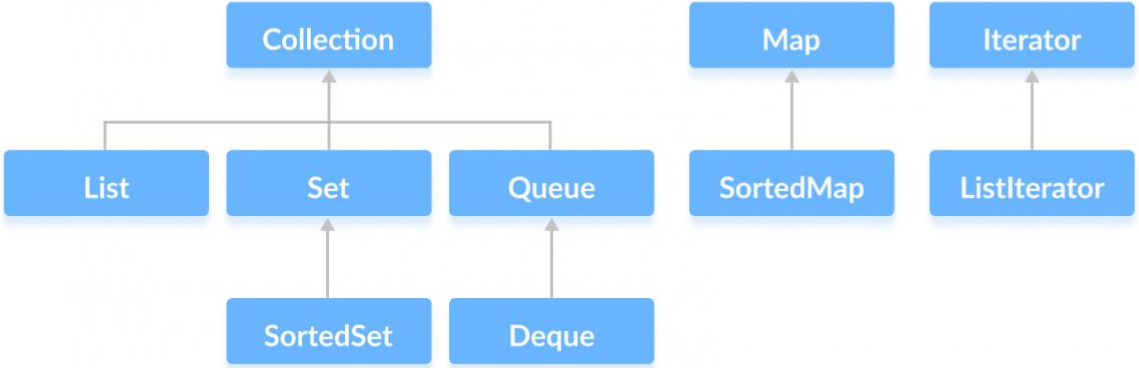


Package

- Organizes classes within libraries
- Creates scope to prevent two classes with same name from overlapping

- java.lang package automatically imported
  - String class, System class, wrapper classes (Boolean, Integer, Double)

Java Collections Framework



List class

A List is:

- Zero-indexed like array
- Ordered set of elements (accessible by index)
- Allows duplicates
- Dynamic in size

- Java List is an interface, so we use ArrayList
    - Called Programming to an Interface
- Must be imported from java.util package

List syntax

List <datatype> objectName = new ArrayList<>();

```
List <String> names = new ArrayList<>();

names.add("Rick");
names.add("Beth");
names.add("Jerry");
names.add(0, “Sam”);

for (int i = 0; i < names.size(); i++) {
  System.out.println(names.get(i));
}
```

- The add method is overloaded – add name is the same, but takes in different parameter listings

List methods

```
List <String> moreNames = new ArrayList<>(Arrays.asList(“Tom”,
“Tim”, “Joe”, “Jim”));

System.out.println(moreNames.size());  // prints 4
moreNames.add(0, “Jane”);
System.out.println(moreNames); // prints out array elements
moreNames.remove(3);          // removes element in pos 3

System.out.println(moreNames.contains(“Tom”));  // prints true

moreNames.removeAll(moreNames);
// removes all elements from ArrayList

System.out.println(moreNames.isEmpty()); // prints true
```

- The add method is overloaded – add name is the same, but takes in different parameter listings

Primitive Wrapper objects

Lists and other collections can only hold objects!

```
List <Integer> ages = new ArrayList<>();

ages.add(29);
ages.add(21);
ages.add(35);
ages.add(32);

for (int i = 0; i < ages.size(); i++) {
  System.out.println(ages.get(i));
}
```

- Wrapper class wraps primitive types so they can be references types
- Autoboxing is process of converting primitive type to reference type (moving from stack to heap)
- Unboxing is moving from heap to stack, converting back to primitive type

Foreach loop

```
List <Integer> ages = new ArrayList<>();

ages.add(29);
ages.add(21);
ages.add(35);
ages.add(32);

for (Integer age: ages) {
  System.out.println(age);
}
```

- Convenience method to iterate through a collection
- Cannot modify contents during iteration
- Useful for when you don’t need the index, just want to go through to each element

Primitive Wrapper Objects

Collections can only hold Reference Types (objects), so to create a Collection, like List, to hold a primitive data type the primitives Wrapper Class must be used.

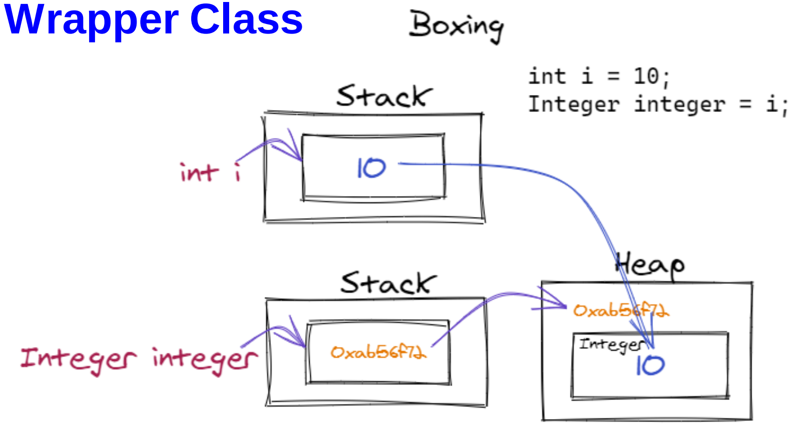
Primitive Type	**Wrapper Class**	**Can be initialized with**
byte	Byte	byte or String
short	Short	short or String
int	Integer	int or String
long	Long	long or String
float	Float	float or String
double	Double	double or String
char	Character	char
boolean	Boolean	boolean or String

Autoboxing

An automatic process of converting between primitive and Wrapper Class data types.

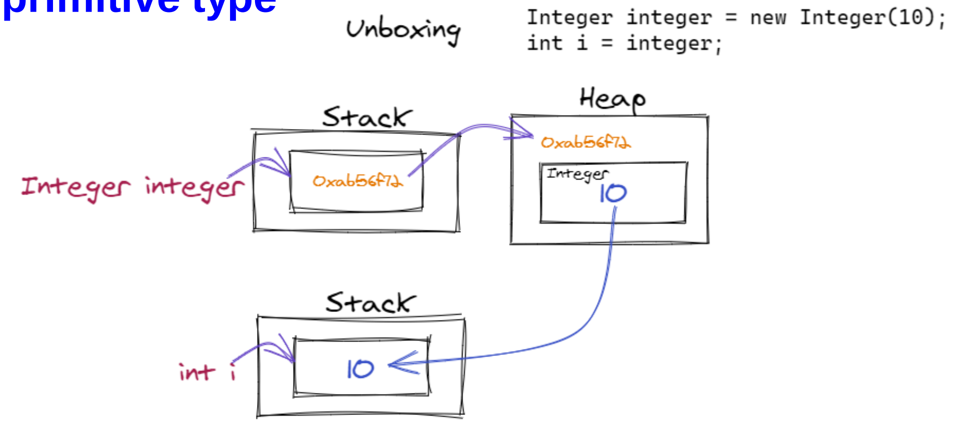
```
Integer i = 10;

Integer x = 20;
int y = x;
x = x + 5;
```

Boxing is moving a *primitive* value from the *Stack* to a *Wrapper Class* object on the *Heap*. **Primitive → Wrapper Class**



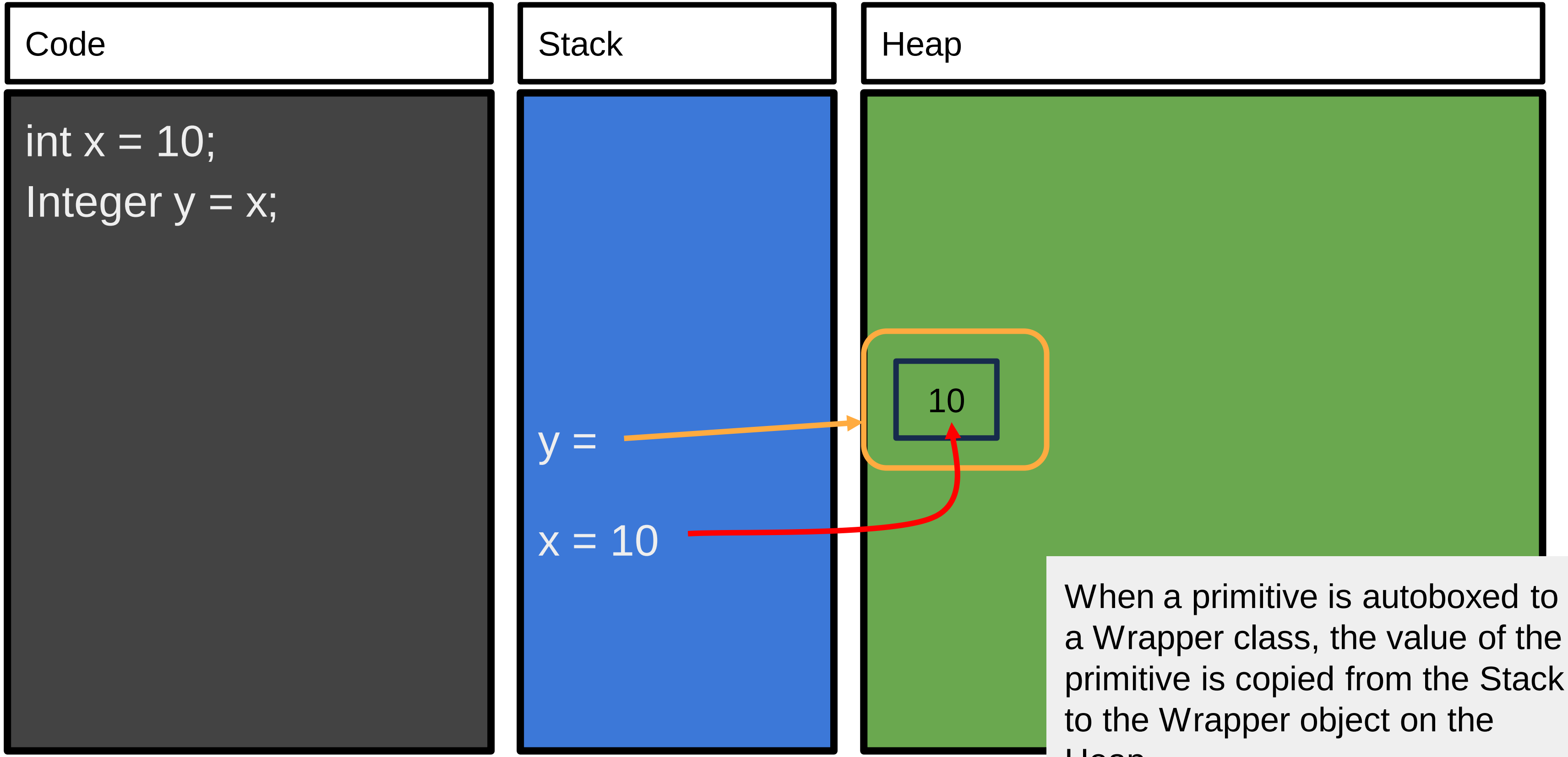
```
int i = 10;
Integer integer = i;
```

Unboxing is moving the value from a *Wrapper Class* on the *Heap* to an appropriate *primitive* value on the *Stack*. **Wrapper Class → primitive type**



```
Integer integer = new Integer(10);
int i = integer;
```

Autoboxing primitive to Wrapper (boxing)



Code

```
int x = 10;
Integer y = x;
```

Stack

y =

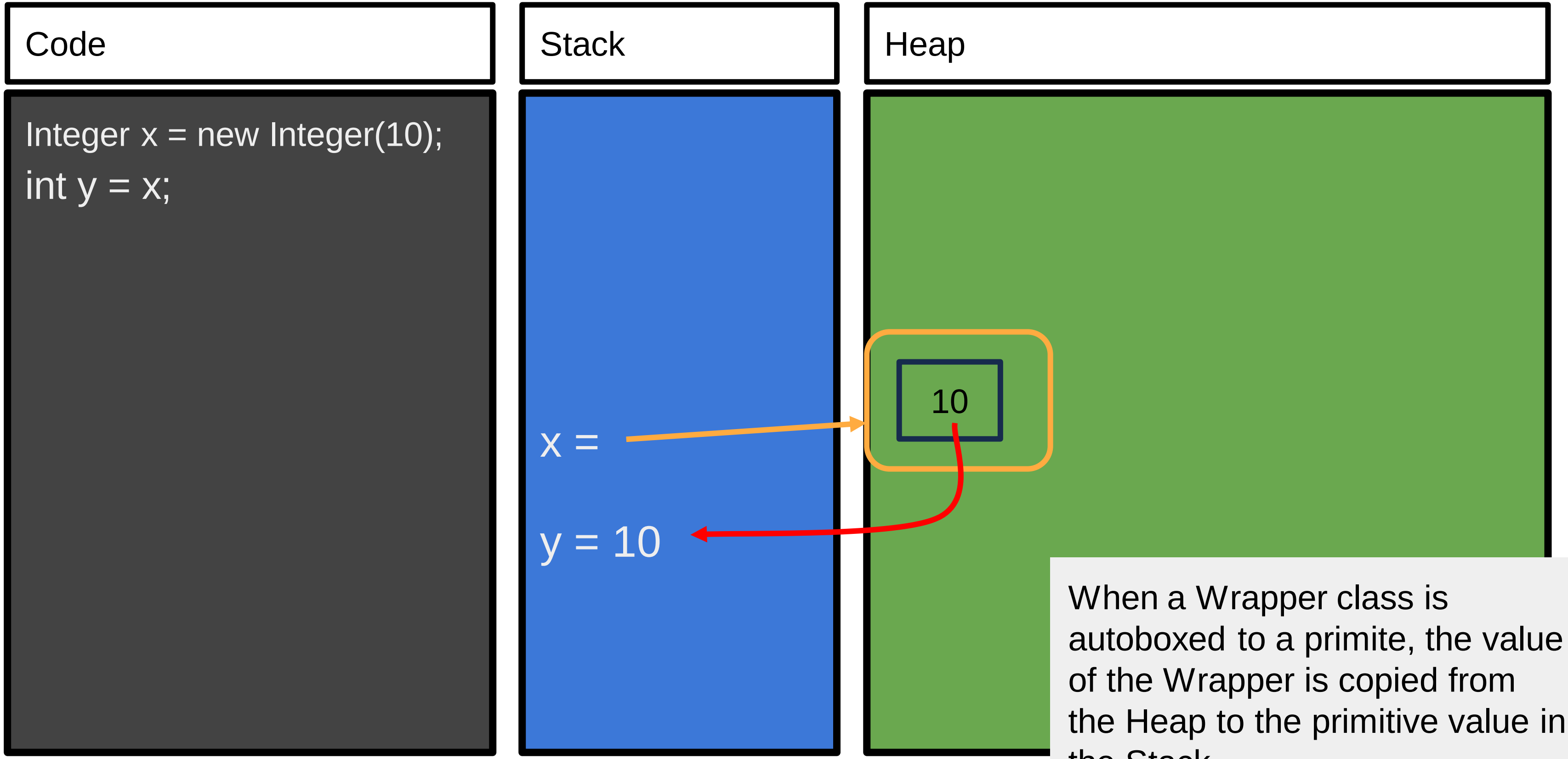
x = 10

Heap

10

When a primitive is autoboxed to a Wrapper class, the value of the primitive is copied from the Stack to the Wrapper object on the Heap.

Autoboxing Wrapper to primitive (unboxing)



```
Integer x = new Integer(10);
int y = x;
```

When a Wrapper class is autoboxed to a primite, the value of the Wrapper is copied from the Heap to the primitive value in the Stack.