Topic 5.1: ArrayLists & Collections

Learning Goals (Week 5):

- Create instances of a built-in Java data type such as an ArrayList.
- Use instances of a built-in Java data type such as an ArrayList.
- Compare and contrast arrays and a Java-defined data type.
- **Use wrapper classes to manipulate primitive types as objects.**

Wrapper classes

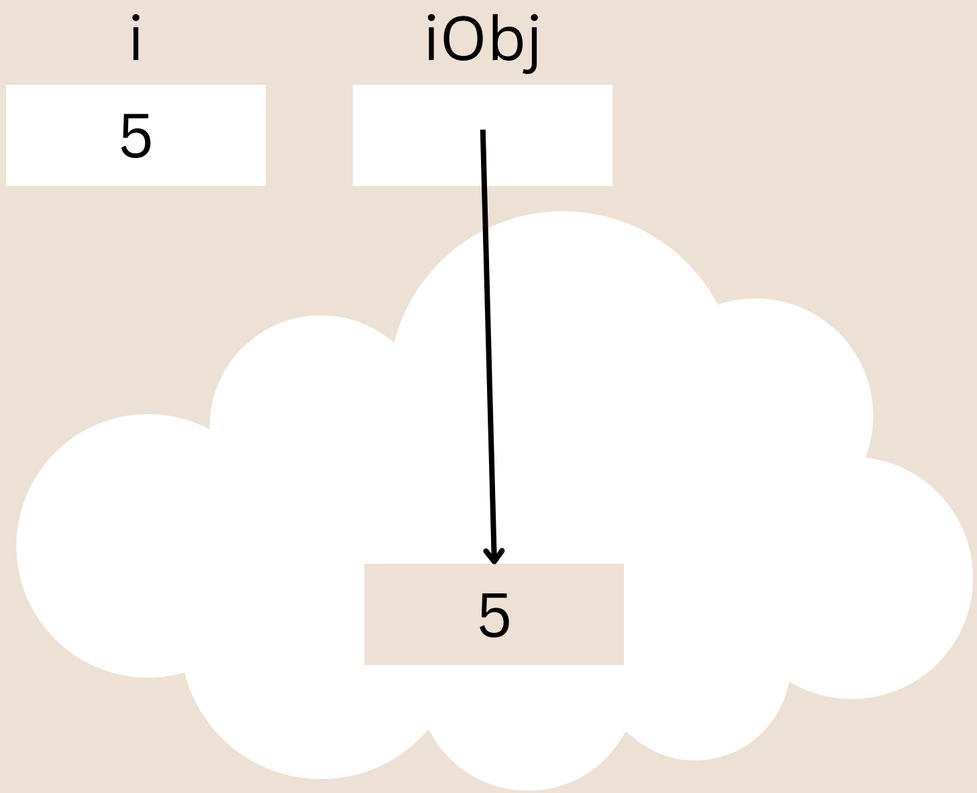
- “Integer object” – a tiny object that contains only a single integer
- Java provides “wrapper classes” for all of the primitive types
  - Primitive types: int, double, boolean, char, long, float, short, byte
  - Object types: Integer, Double, Boolean, Character, Long, Float, Short, Byte
- These store **references** to **immutable** objects (just like **String**)

Wrapper classes

```java
// Create variables
int i;
Integer iObj;

//Assign values
 i = 5;
iObj = new Integer(5);

//Use the values
System.out.println(i+1);
System.out.println(iObj.intValue( )+1);

//Change the values
i=3;
iObj = new Integer(3);
```

Wrapper classes

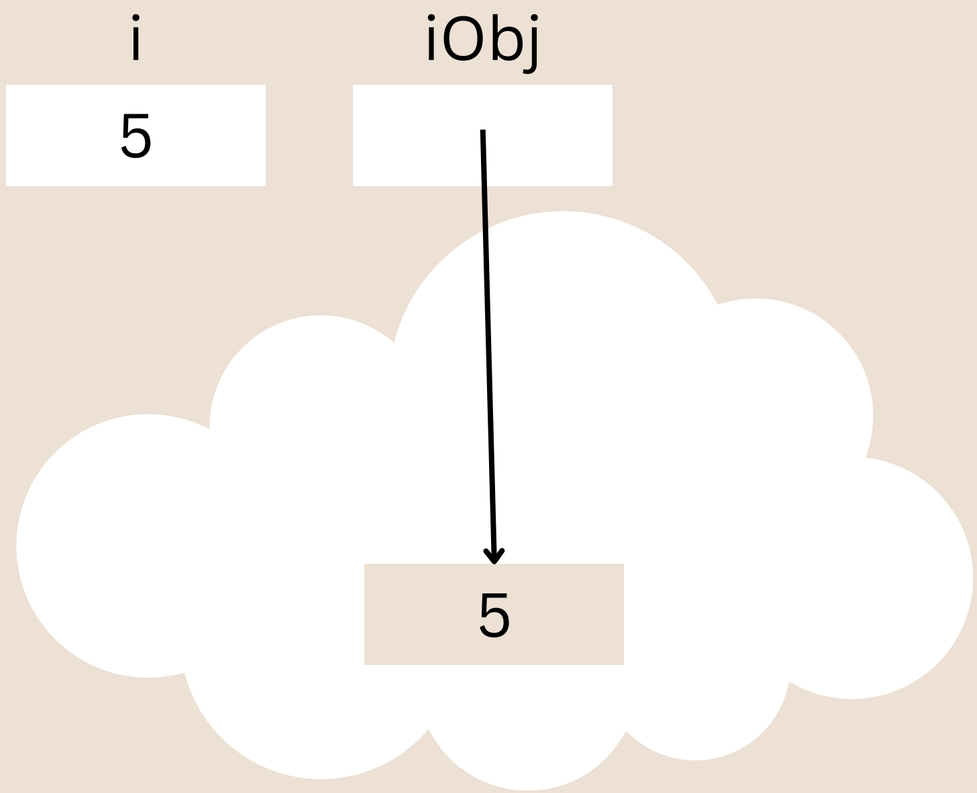
```java
// Create variables
int i;
Integer iObj;

//Assign values
 i = 5;
iObj = new Integer(5);

//Use the values
System.out.println(i+1);
System.out.println(iObj.intValue( )+1);

//Change the values
i=3;
iObj = new Integer(3);
```



Wrapper classes

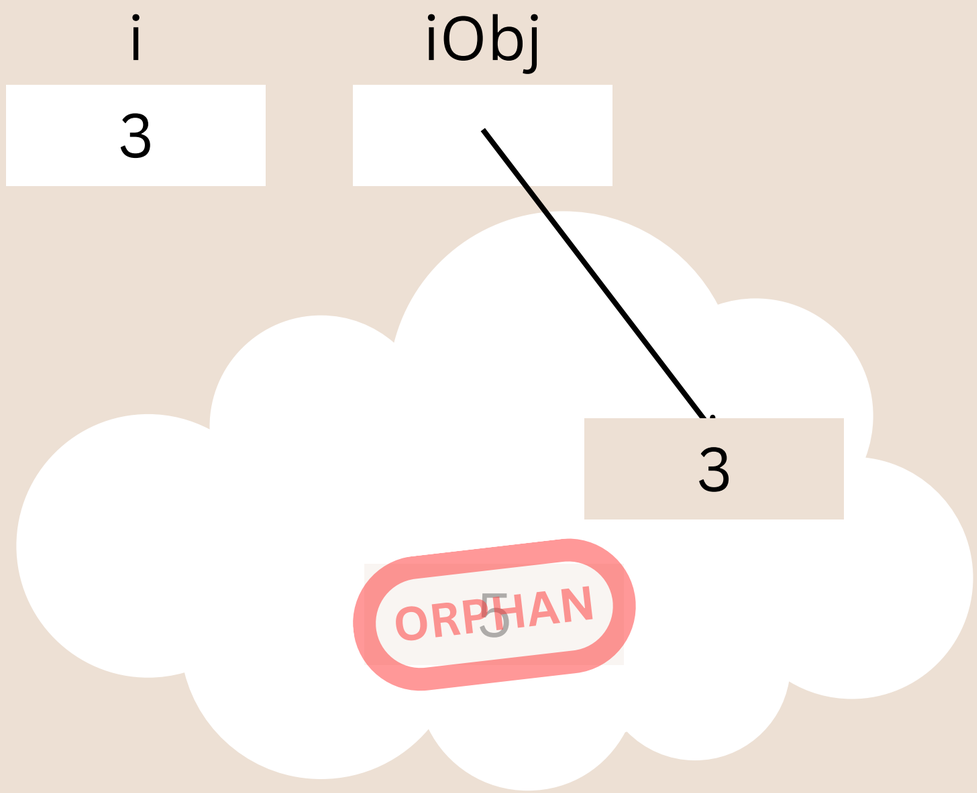
```java
// Create variables
int i;
Integer iObj;

//Assign values
 i = 5;
iObj = new Integer(5);

//Use the values
System.out.println(i+1);
System.out.println(iObj.intValue( )+1);

//Change the values
i=3;
iObj = new Integer(3);
```

print 6s



Wrapper classes

```java
// Create variables
int i;
Integer iObj;

//Assign values
 i = 5;
iObj = new Integer(5);

//Use the values
System.out.println(i+1);
System.out.println(iObj.intValue( )+1);

//Change the values
i=3;
iObj = new Integer(3);
```



Type casting

- Using wrappers that way is clumsy… (the new keyword everywhere)
- Java (since SE5 – very old now) will convert freely between primitive types and their wrapper type
  - you do not have to specify new, .intValue, etc

- If it is expecting an int value, and you use an Integer, it will “**unbox**” it (extract the value from it)

- If it is expecting an Integer value, and you use an int, it will “**box**” it (create an Integer with that value)

Type casting

```
int x = new Integer(34); //silly, but legal
Integer y = 45;          //also legal.
x = y;                   //OK. It extracts its value.
y = x;                   //OK. It creates an object for you.
```

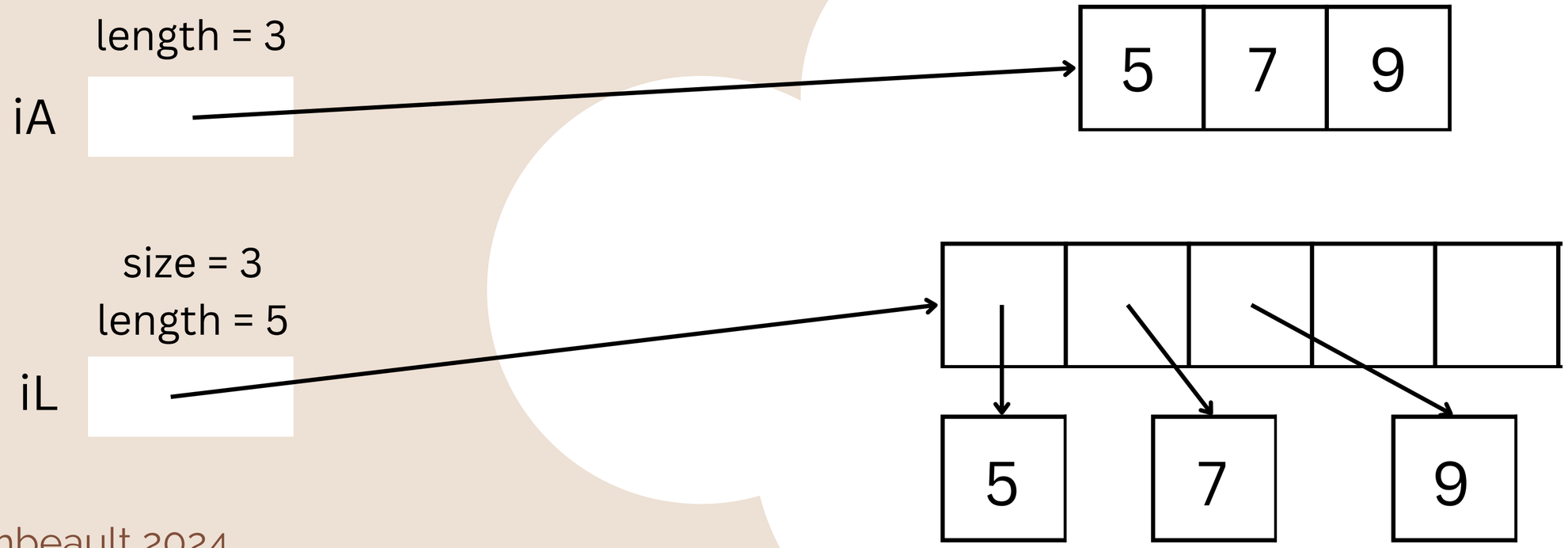
Primitives vs Objects (Implicit Boxing/Unboxing)

```
// Create variables
int i;
Integer iObj;

// Assign values
i = 5;
iObj = 5;  // boxes

// Use the values
System.out.println(i+1);
System.out.println(iObj+1 ); // unboxes

// Change the values
i=3;
iObj = 3;  // boxes
```

- Works just like before (under the hood)
- User side is just cleaner
- Use int and Integer the same but now Integer class lets us have an ArrayList of Integers

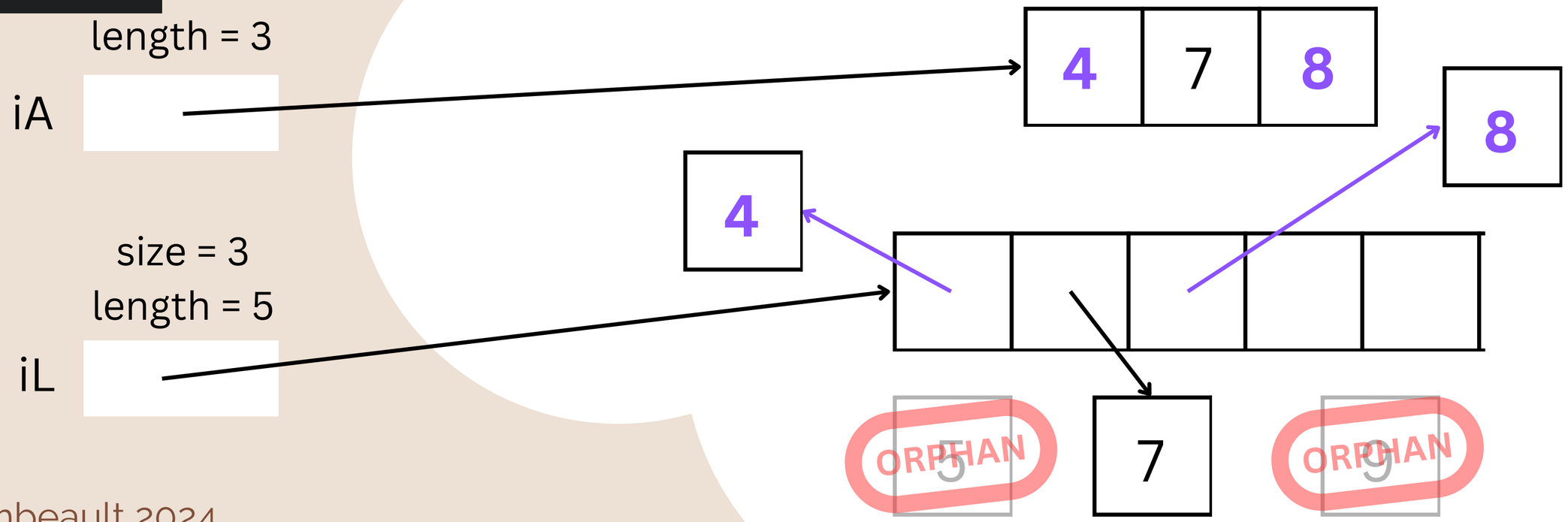
int[ ] vs ArrayList<Integer>: Storage

```
int[ ] iA = {5,7,9};
ArrayList<Integer> iL = new ArrayList<Integer>();
iL.add(5);
iL.add(7);
iL.add(9);
```



int[ ] vs ArrayList<Integer>: Storage

```
iA[0]=4;
iA[2]=8;


iL.set(0,4);
iL.set(2,8);
```



int[] vs ArrayList<Integer>

- ArrayLists are convenient and give many useful methods to handle lists which freely grow and shrink
- But, as the last slide shows, there is internal complexity, which translates to a speed penalty
- Usually OK, since modern processors are very fast (you wouldn't notice the difference on your computers in this class)
- But in computationally intensive tasks, ordinary arrays would be preferred

A note on the power of ArrayLists

- An ArrayList<Object>, or just ArrayList, is flexible and powerful

- It can store a list of any kind of data
  - With a mixture of types

- This is how dynamic interpreted languages do just about everything
- More to come on this in the coming weeks and future courses!
- **HOWEVER…**
- When you get( ) an element, you just get an Object
  - You probably can't do anything with it until you (down)cast it to the correct type
  - And you probably need to check **instanceof** before doing the cast, to do it safely (we won’t see more of this keyword right now, but it does exist!)

Pause & Practice

- Go back to the arrays practice from the previous lectures and rewrite those problems using ArrayLists

- This allows you to practice ArrayList use with code you should already feel comfortable with