Pulse Survey is open

Review Questions

What is a class?

What is an object?

How is a class different from an object?

Give an example of a primitive type variable and a reference type variable?

When would a NullPointerException occur?

Review Questions 2

What, if anything, is wrong with the following code?

```
if( language == “java” ){
  // Code
}
```

In the code below, what is:

- the variable’s type?
- the variable name?
- the part of code that creates an object?

```
Scanner scan = new Scanner(System.in);
```

Collections Part 1

Module 01 - 07

Today’s Objectives

1. Packages
2. BigDecimal
3. LocalDate
4. Collections
5. List<T>

Packages

1. **Packages** organize Java *classes* and creates scope to prevent two classes with the same name from having a *name collision* (i.e. overlapping in a way that makes them ambiguous)
2. Creates a **namespace**
   a. Groups related elements
   b. Everything in the group must have a unique name within that group
3. Class names must be unique in the package, but not across packages

    com.abank.accounts.deposit.CheckingAccount

    com.techelevator.exercises.CheckingAccount

Fully Qualified Class Name

Classes can be referred to by the Class name when there is an *import* in the class.

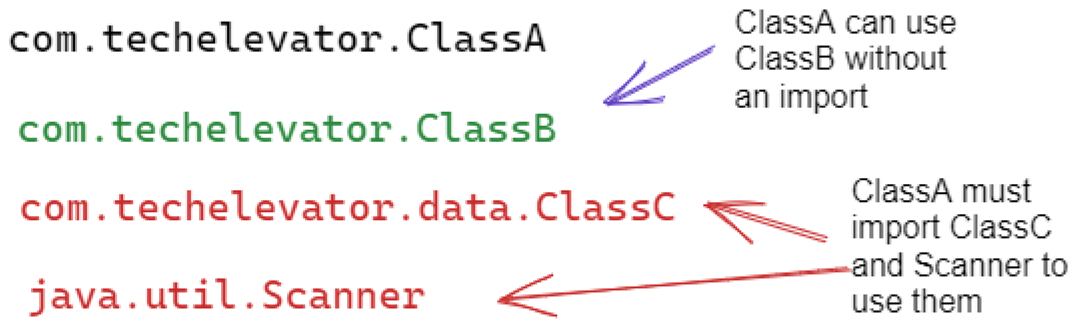
```
import java.util.Scanner.

Scanner in = new Scanner(System.in);
```

Or can be referred to by the fully qualified class name, which is the package name along with the class name.

```
java.util.Scanner in = new java.util.Scanner(System.in);
```

Importing Classes

Classes in the same package can see each other with an import.


com.techelevator.ClassA
com.techelevator.ClassB
com.techelevator.data.ClassC
java.util.Scanner
ClassA can use ClassB without an import
ClassA must import ClassC and Scanner to use them

The **java.lang** package is a special case, and can be used in any class without needing to explicitly import it. We must explicitly import all other classes, or use their fully qualified class name. java.lang contains core language features like String.

A * can be used to import all classes from a package.

```
java.util.*;
```

This is can lead to unexpected behaviors so it often discouraged outside of testing.

BigDecimal

java.math.BigDecimal
https://docs.oracle.com/javase/7/docs/api/java/math/BigDecimal.html

Does not have a floating point rounding problem like double and float.   Does not truncate like integer.   Is commonly used for currency and other calculations that require a high and precise significance of precision.

import java.math.BigDecimal;

BigDecimal amount = new BigDecimal(<value>);   ← CANNOT use a No-Argument Constructor

Can't use operators +, - , %, /, *, <, etc. instead use methods
example: amount.add()

BigDecimal is immutable.

```
BigDecimal amountOne = new BigDecimal(100.50);
BigDecimal amountTwo = new BigDecimal(200.25);
BigDecimal combinedAmount = amountOne.add(amountTwo);
```

In the above code, when add is called the value of amountOne is not changed, it remains 100.50. Instead a new BigDecimal is returned with the sum (300.75). This is due to BigDecimal being immutable, and is the same as when you use a String function like substring() or toUpperCase()

LocalDate

java.time.LocalDate
https://docs.oracle.com/javase/8/docs/api/java/time/LocalDate.html

Provides a standard way to store and work with Dates in Java.

```
// Today's Date
LocalDate today = LocalDate.now();
```

java.time.DateTimeFormatter can be used with LocalDate to provide the format of the date for printing or input.

```
DateTimeFormatter dateFormatter = DateTimeFormatter.ofPattern("MM/dd/yyyy");

LocalDate graduationDay = LocalDate.parse("08/19/2022", dateFormatter);
```

java.util.Date is often seen in online examples, it is an older way of working with dates in Java. It is outdated and difficult to work with.

Arrays Recap

Arrays are simple data structures that allow us to work with a collection of like data.

- Arrays are fixed sized
- Arrays are indexed starting at 0
- Arrays are accessed using [] brackets and the index

```
instructors[1]
```

String[] instructors

Index	Value
0	John
1	Rachelle
2	Steve
3	Matt
4	Dan

Collections

Collections are classes in the java.util package that define data structures that make working with a set of data easier.

Collections are written to be generic so that they are useful for all reference types.

CollectionType<T> ← <T> indicates a generic and that the Collection type can hold any reference type.

java.util.Collection JavaDoc

Array and List Operations

	Array	List
Declare	✓	✓
Initialize	✓	✓
Get element	✓	✓
Set element	✓	✓
Iterate (loop)	✓	✓
Add		✓
Remove		✓
Insert		✓

List<T>

1. An ordered set of elements that is 0-indexed like an array.
2. Maintains the Order of Insertion
3. Elements can be accessed by index
4. Allows duplicate elements
5. A List can hold 1 data type, like an Array, that is defined by <T>
6. Can grow and shrink as items are added and removed

Instantiating a List<T>

```
List<DataType> variableName = new ArrayList<DataType>();
```

List is an ***interface***, which unlike a class contains no code and just creates a data type with guaranteed functionality.

<DataType> sets the Reference Data Type that the list will hold.

ArrayList is a ***class that implements*** the **List** data type, and will be instantiated by the new keyword. The ArrayList will hold the data type given, and the data type must match the one given for the variable created with the List interface.

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

List methods

size()	Returns the number of elements in the list
add(x)	Adds an element to the bottom of the List
add(index, x)	Adds an element to the List at the given index.
get(index)	Returns the element at the given index
remove(index)	Removes the element at the given index

A List can be converted to an Array:

```
            String[] instructorsArray = instructors.toArray( new String[ instructors.size()
] );
```

An Array can be converted to a List:

```
            List<String> instructorsList = Arrays.asList( instructorsArray );
```