Lecture Notes

EL Armaments

Arrays and ArrayLists

Arrays in Java

Arrays: An array is a collection of elements of similar data types that are stored in contiguous memory locations. The length of an array is fixed, and only a fixed set of elements can be stored in a Java array.

The first index of an array is always 0.

Advantages of a Java Array

1. It is simple to create.
2. It is easy to access the elements within a Java array.

Disadvantages of a Java Array

1. Only a fixed number of elements can be stored in a Java array, and the length of the array cannot be increased or decreased.

Every time you create a new array (for example, 'random'), you also **create a new array object**. Depending on the data type of that particular array object, you can store either an int, double, string or any primitive data type/object in the array. You can also declare any variable as an array in the following ways:

```
int random[]
int[] random
```

Click on this link to learn how to declare an array of different data types.

ArrayList

A Java ArrayList is quite similar to an array and has the additional capability of adding or removing elements dynamically at runtime. This is why **ArrayList is also referred to as a dynamic array**. It is used to store a group of elements and allows you to store duplicates.

```
ArrayList studentList = new ArrayList();
```

In ArrayList, **all the elements are of the data type Object**, which needs to be typecast in order to be accessible.

```
public static void printStudentList(ArrayList students) {
  for(Object o : students) {
    Student s = (Student) o;
    System.out.println(s.getDetails());
  }
}
```

Type Safety

Type safety is a property of programming languages. Type safety means that a well-written program with no syntax errors detected at the compile-time should not throw a runtime error. You can learn more about type safety by referring to this additional reading material.

Now, **how could a single ArrayList store both 'String' and 'Student' data types?**

Let's revise the concept of inheritance to understand this better.

An object of a child class can be stored into a variable of a parent class. For example, if there are two classes, 'Cat' and 'Dog', which inherit the parent class 'Pet', then an object of 'Cat' class can be stored in a variable of type 'Pet', as shown below.

```
Pet ob1 = new Cat();
Pet ob2 = new Dog();
```

An ArrayList stores all elements of the data type 'Object', which is the parent class of all Java classes by default.

As 'Object' class is the parent class of both 'String' class and 'Student' class, you are able to store these data types in a single ArrayList.

Generally, **you should create an ArrayList of the same data type.** However, you may encounter runtime errors if you accidentally store an element that cannot be cast.

To deal with this error, you can define the ArrayList using generics.

ArrayList Using Generics

Defining ArrayLists using generics ensures that only the data of a particular data type can be stored in the ArrayList. For example, consider the following two lines of code:

```
ArrayList<String> names = new ArrayList<String>();
ArrayList<Student> studentList = new ArrayList<Student>();
```

In the 'names' ArrayList, only the data of data type 'String' can be stored. Similarly, in the 'studentList' ArrayList, only the data of data type 'Student' can be stored. This special ArrayList, in which you **cannot add elements** of different data types, is referred to as an **ArrayList using Generics**. If you do try to add elements of any other data type in this ArrayList, it will throw a compile-time error. Some of the advantages of using Generics are as follows.

1. The ArrayList can only hold a specific data type, and the elements of other data types are not allowed.
2. Typecasting is not required.
3. There is a conversion of potential runtime errors into compile-time errors.

Format to Declare ArrayList Using Generics

The class ArrayList can be declared using generics in the following method:

```
ArrayList<datatype> listName = new ArrayList<datatype>();
```

Here, the data type to be mentioned is always non-primitive (reference). For example, Student and String are declared as classes. Primitive data types such as int, char and double cannot be used here.

If you want to store primitive data types in ArrayList classes, you will need to use their Object cousins, which are Integer, Double, Float and Boolean.

You can create the Generics ArrayList with elements of certain data types using the following formats:

1. ArrayList of int-type values

```
ArrayList<Integer> list = new ArrayList<Integer>();
```

2. ArrayList of double-type values

```
ArrayList<Double> list = new ArrayList<Double>();
```

3. ArrayList of float-type values

```
ArrayList<Float> list = new ArrayList<Float>();
```

Operations on ArrayList

The following basic operations can be performed on an ArrayList:

1. Adding an element at any arbitrary position
2. Removing an element
3. Searching for an element

The following methods can be used to **add elements** to the ArrayList:

1. **add(Object o):** This method appends the specified object ‘o’ to the end of ArrayList. Its return type is Boolean, which returns TRUE when the element is added to the list.
2. **add(int index, Object o):** This method inserts a specified object into a specified position in the ArrayList.

The following methods can be used to **remove elements** from the ArrayList:

remove(int index): This method is used to remove an element from the ArrayList at the specified index.

clear(): This method is used to delete/remove all the elements from the list, as shown in the example of the syntax of this method below.

```
studentList.clear();
```

The following method can be used to **search for an element** in the ArrayList:

contains(Object o): This method searches for the element in the ArrayList and returns ‘true’ if the element is present in the list.

Additional Reading

You can go through the Method Summary and Method Detail tables available on this page to learn more about other methods in the ArrayList class. You can use these methods to perform different operations.

LinkedLists (Optional)

A LinkedList is another data structure in Java, which works like an ArrayList and has similar methods and functionalities.

```
LinkedList<Student> studentList = new LinkedList<Student>();
```

However, **in a LinkedList the elements are not stored in contiguous memory locations.** This affects programs in terms of performance, which means even though the outputs of all the operations are the same, the speed at which these operations are conducted is different.

All the three methods, add, remove and contain, work in the same way in a linked list as they do in the ArrayList.

A LinkedList has another method that is used to access an element from a list through its index, which is as follows

get(int index): This method returns the element at the specified position in the list.

You can learn how this method works from this link.

Lists and Polymorphism

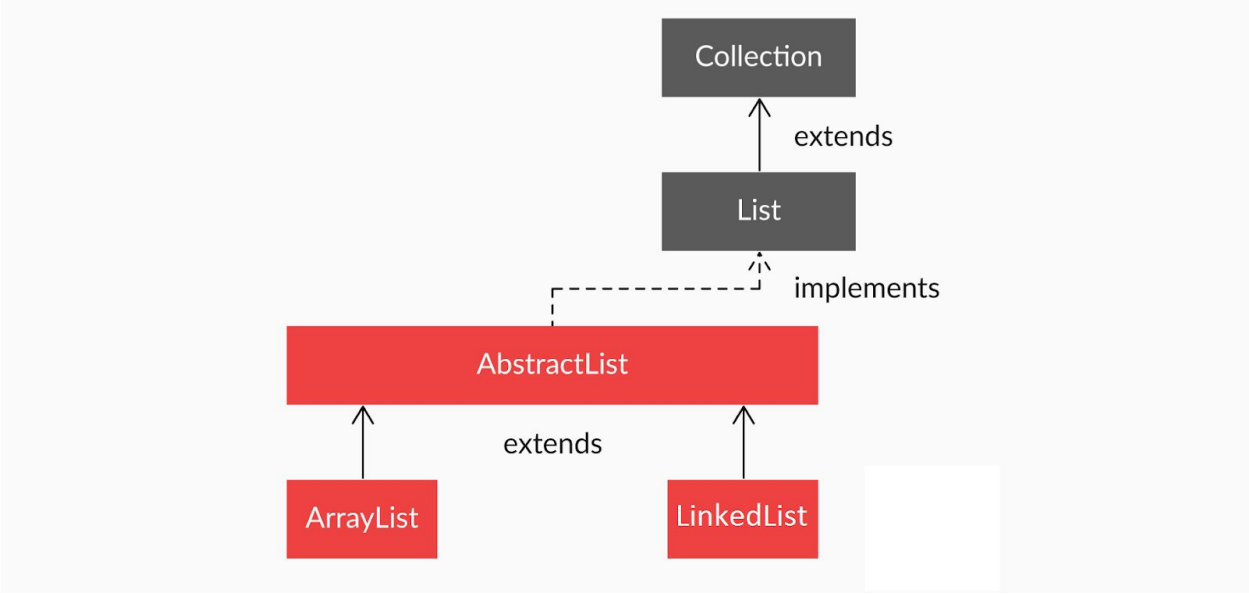
A **List** is an interface that is implemented by the **ArrayList** and **LinkedList** classes. This is why you can instantiate both 'ArrayList' and 'LinkedList' by declaring the type of the variable as List.

```
List<Student> studentList1 = new ArrayList<Student>();
List<Student> studentList2 = new LinkedList<Student>();
```

Given below is the printStudentList as a polymorphic function. It uses List instead of ArrayList or LinkedList, thus inter-operates smoothly with both types.

```
public static void printStudentList(List<Student> students) {
  for(Student s : students) {
    System.out.println(s.getDetails());
  }
}
```

You can refer to the diagram given below to understand how these classes and interfaces are linked to a larger interface called **Collection**.



1. The List interface extends the Collection interface, or in other words, List is the child interface of Collection.
2. AbstractList implements the List interface, which is further extended by the ArrayList and LinkedList classes. In other words, the ArrayList and LinkedList classes are implementations of the List interface.
3. The AbstractList class is extended by the ArrayList and LinkedList classes. In other words, ArrayList and LinkedList are the subclasses of the Abstract class.

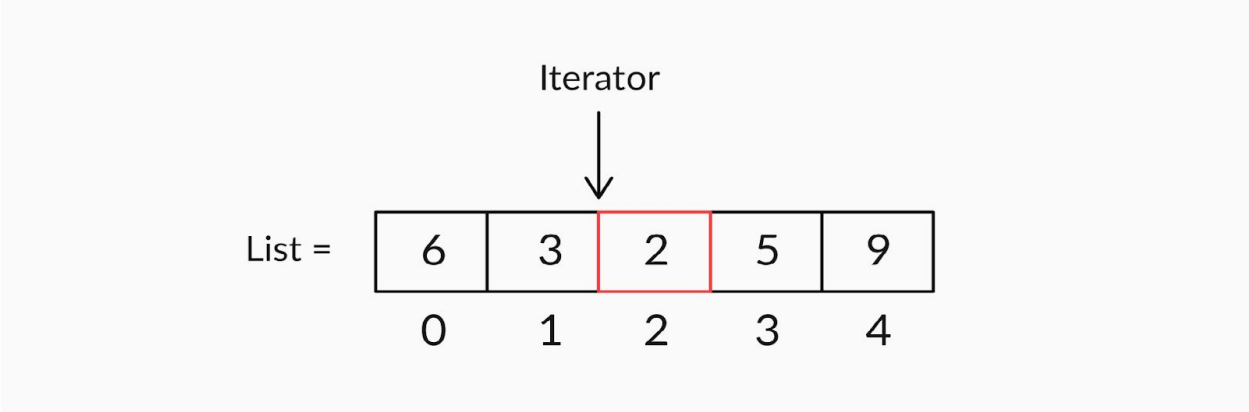
ListIterator

An iterator is an interface in Java, which is used to iterate over a collection of objects. In simple terms, an 'iterator' acts as a cursor to an element in a collection. You can also use the 'iterator' (or the cursor) to move to the next element in the collection.

```
ListIterator<Student> it = students.listIterator();
```

ListIterator is a subinterface of the Iterator interface, which is used to iterate over a list. It has a lot more features than the Iterator interface. Some of these features are as follows:

1. ListIterator is used to traverse a list in any direction, i.e., forward or backward, quite easily.
2. It does not point to any current element; its cursor position always lies between the previous and next elements (See Figure given below).
3. It has methods that can be used to determine whether a next or previous element is present in a list. You can also use the methods to find the value of the next or previous element, as well as its indices.



Iterating Forward Using ListIterator:

```
private static void iterateFwd(List<Student> students) {
  ListIterator<Student> it = students.listIterator();
  while(it.hasNext()) {
    System.out.println(it.next().getDetails());
  }
}
```

Iterating Backward Using ListIterator:

```
private static void iterateBkwd(List<Student> students) {
  ListIterator<Student> it = students.listIterator(students.size());
  while(it.hasPrevious()) {
    System.out.println(it.previous().getDetails());
  }
}
```

Additional Reading

Read this to know more about the additional methods of the listIterator interface.

Annotations

Annotations can be used to add some extra information about a piece of code. Such information can be used by other tools to treat that piece of code differently. You will be using a lot of annotations when you develop your own application.

All annotations in Java always start with '@'. For example, @Override.

@Override Annotation

What is the use of the @Override annotation?

When a method is marked with the @Override annotation, the compiler searches for that particular method in the parent class. If the method is not found in the parent class, then the compiler throws an error.

Advantages of using the @Override annotation

- Fewer chances of bugs (because of an incorrect method name)
- Better readability

@Deprecated Annotation

What is the use of the @Deprecated annotation?

The @Deprecated annotation is used to mark a piece of code that should not be used to write new code. If your code includes the deprecated code, the compiler will show a warning.

Advantages of using the @Deprecated annotation

- Warns the team/programmer about deprecated elements
- Helps in maintaining the hygiene of code

Additional Reading

Refer to this link to learn more about Custom Annotations.

Introduction to Version Control and Git

Version Control

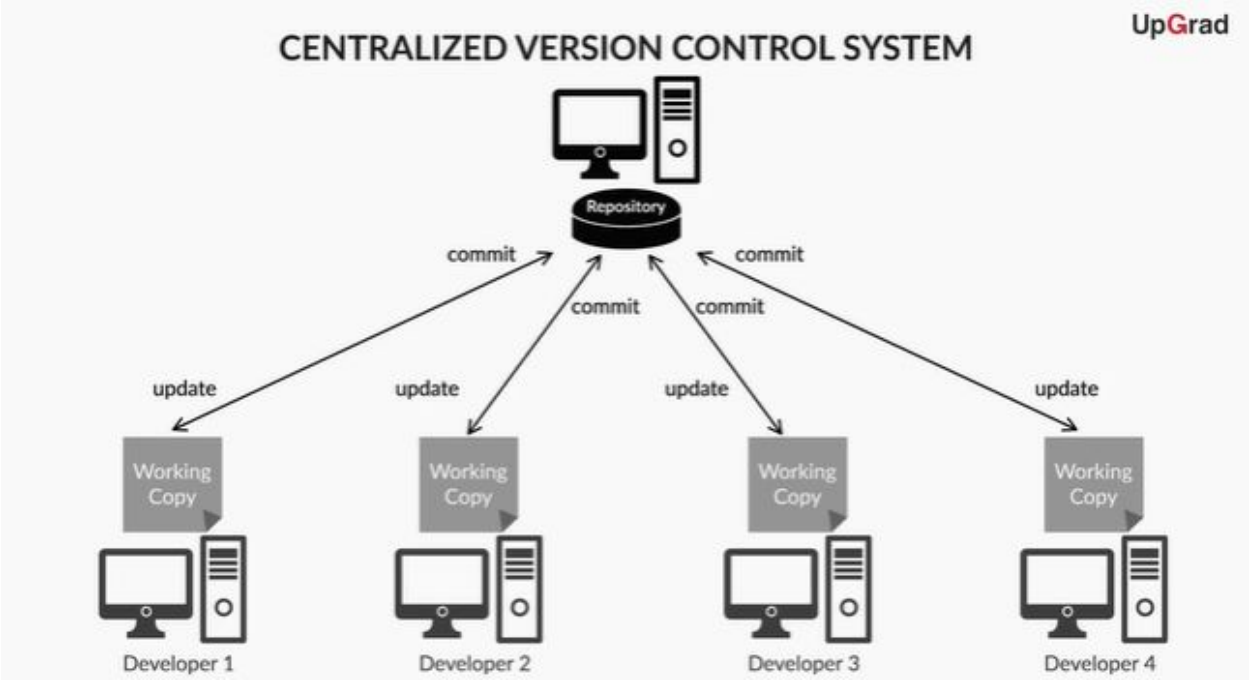
Version Control is a system that records or keeps track of the changes made to a file or a set of files over time so that you can recall or revert to specific versions later.

Types of Version Control Systems

Version control systems are of the following two types:

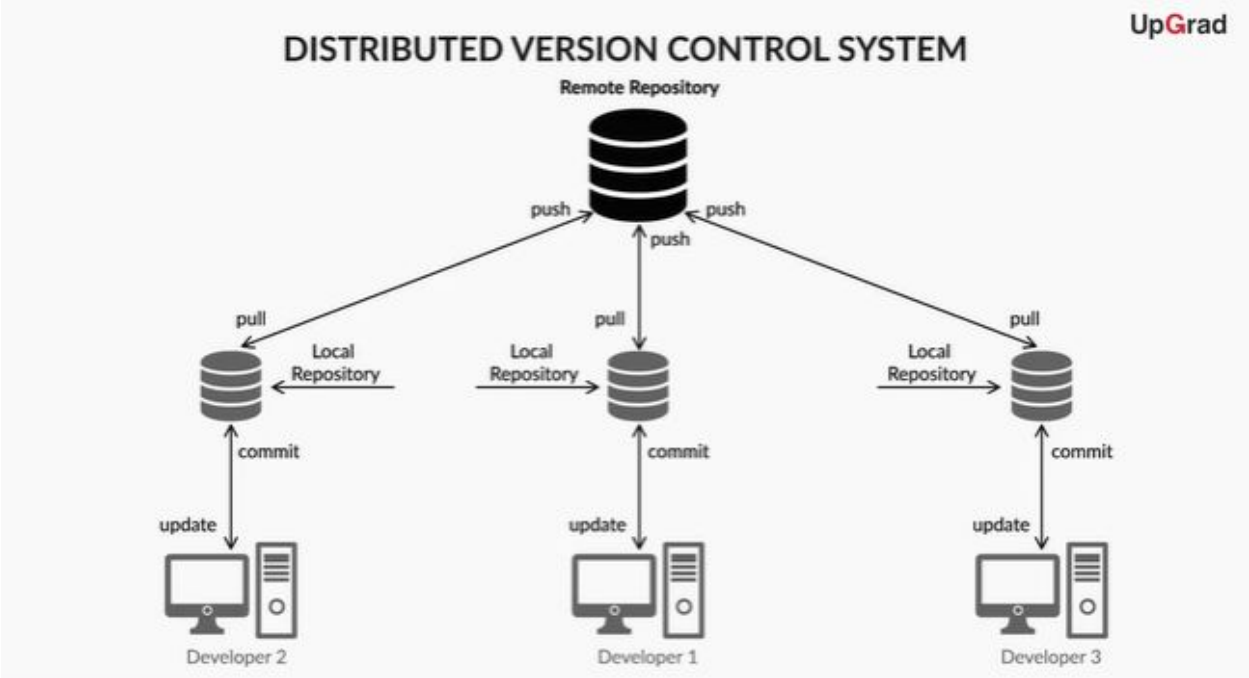
Centralized Version Control System

Centralized version control system works as a client-server model. This system has one centralized server and a localized repository file system that is accessible by many clients.



Distributed Version Control System

In the distributed version control system model, all developers have their own local file system, and changes between different file systems are implemented locally on their machines.



Git and GitHub

Git and GitHub are different.

Git is a **distributed version control system**. It is a tool that is used to manage your project source code history.

GitHub is a **web-based git file hosting service** that enables you to showcase or share your projects and files with others.

Repository: A repository is a directory that contains your project work. All the files in the repository can be uploaded to GitHub and shared with other people, either publicly or privately.

The three steps that your files may go through internally are as follows:

1. Modified
2. Staged
3. Committed

Basic git commands: Click here to read about some of the basic git commands.

- First-time git set-up
  - For the first-time git configuration, you need to use the following commands:
    - *git config --global user.name "random"*
      - This command will be used to enter your GitHub username.
    - *git config --global user.email "random@example.com"*
      - This command will be used to enter your GitHub username.

- Making a commit and pushing your changes to GitHub
  - Use the following commands to do this:
    - git init
    - git add filename
    - git commit -m "commit message"
    - git remote add origin <url of the remote repository>
    - git push -u origin master

- The following git commands that are used quite frequently:
  - **git status**: This command will display the state of the working directory and the staging area. In other words, it allows you to view the changes that have been staged and the changes that have not been added to the staging area.
  - **git log**: This command shows you the commit details. It lists out the commits made in the repository in reverse-chronological order, i.e., the most recent commits show up first followed by other commits. It shows commits along with the following details:
    - The commit ID or SHA

- Author's name (who made the commit)
- Date and time (when the commit was made)
- Commit message

The 'git remote add origin url' command

- You can use this command to add a new remote repository to your local repository. To do so, you should use the '**git remote add'** command on the terminal, in the directory where your repository is stored. The **git remote add** command takes the following two arguments:
  - A remote name, for example, origin (it can be any name)
  - A remote URL, for example, https://github.com/user/repo.git (the address of the repository on your GitHub account to which you want to link your local repository.)

Pushing changes after the first commit

When you are pushing your changes after the first commit, i.e., after:

1. Initialising your git repository, and
2. Linking it with the remote repository on git

you only need to run the following git commands to commit any changes into your local git repository and then push those changes to the remote repository on GitHub:

1. **'git status'**: This command is used to check which files are changed and not yet moved to the staging area.
2. **'git add <filename>' or 'git add .'**: This command is used to add modified files to the staging area. You can add a specific file using the command 'git add <filename>'. If you want to add all the modified and unstaged files present in the workspace to the staging area, you can use the 'git add .' command.
3. **git commit -m** "**New commit message**": This command gives a new commit message and commits all the files present in the staging area.
4. **git push -u origin master or git push**: This command is used to upload all the files and changes that were included in the most recent commit, to your remote repository on GitHub.

Downloading Repository

git clone

This command is used to clone or copy another user's repository. When you start at a new job, you will most likely work on existing projects. In order to get the code of that project in your system, you will need to clone it from an existing GitHub repository.

The syntax for the command is as follows:

git clone <url of the remote repository>

Example:

git clone https://github.com/user/repo.git

You can go through this link to learn more about this command.