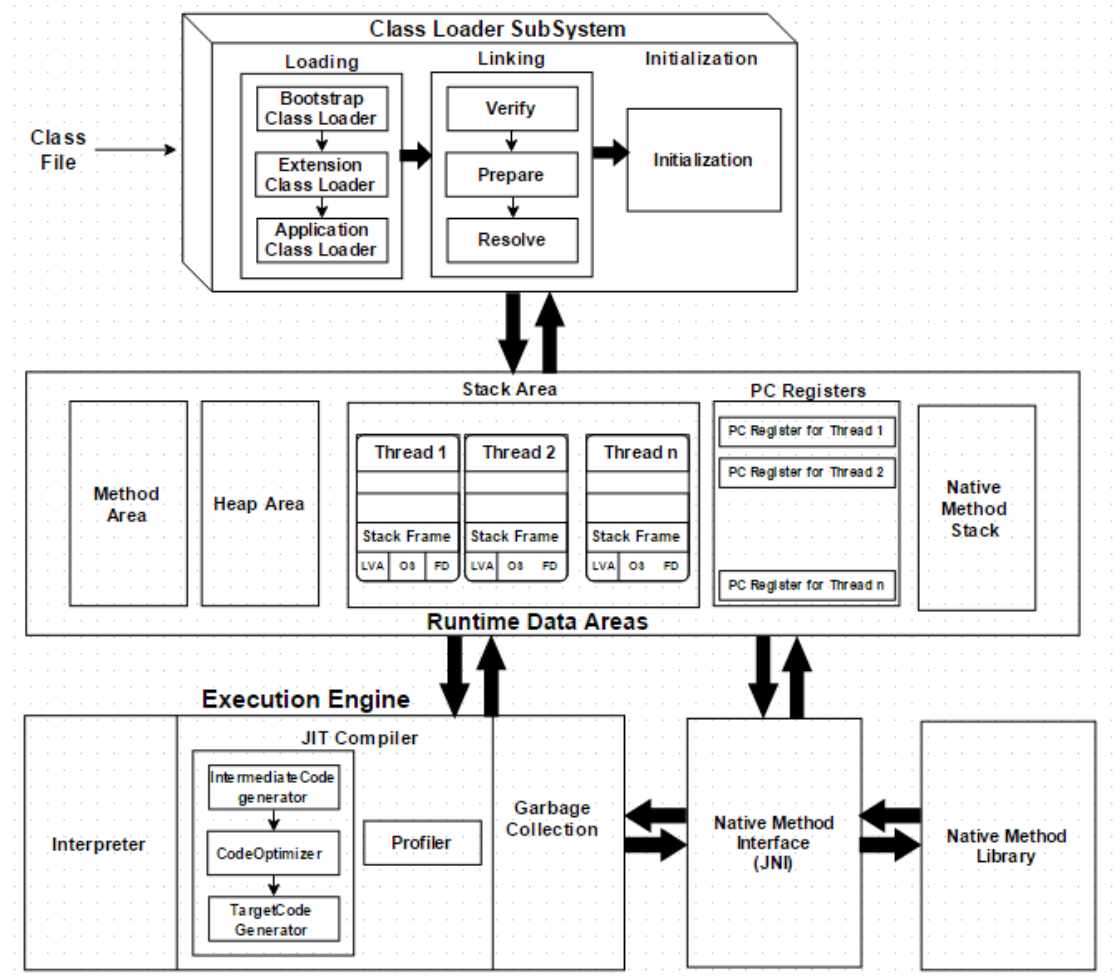
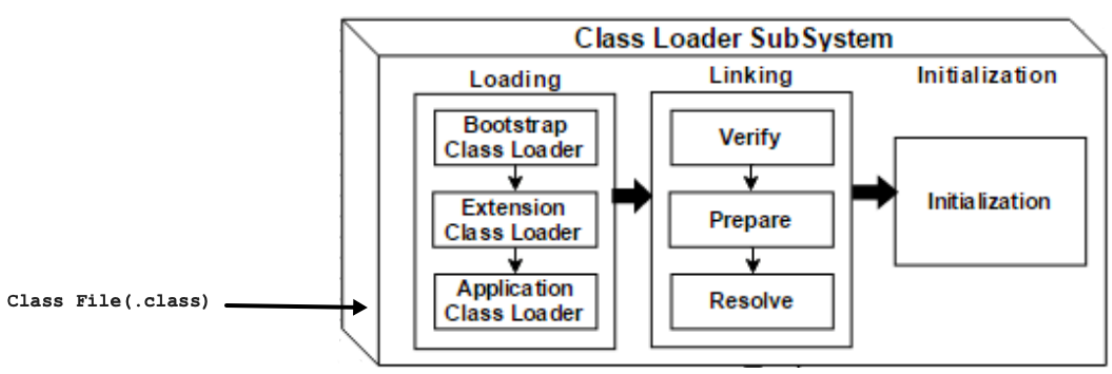
Java Language Features

JVM Architecture:



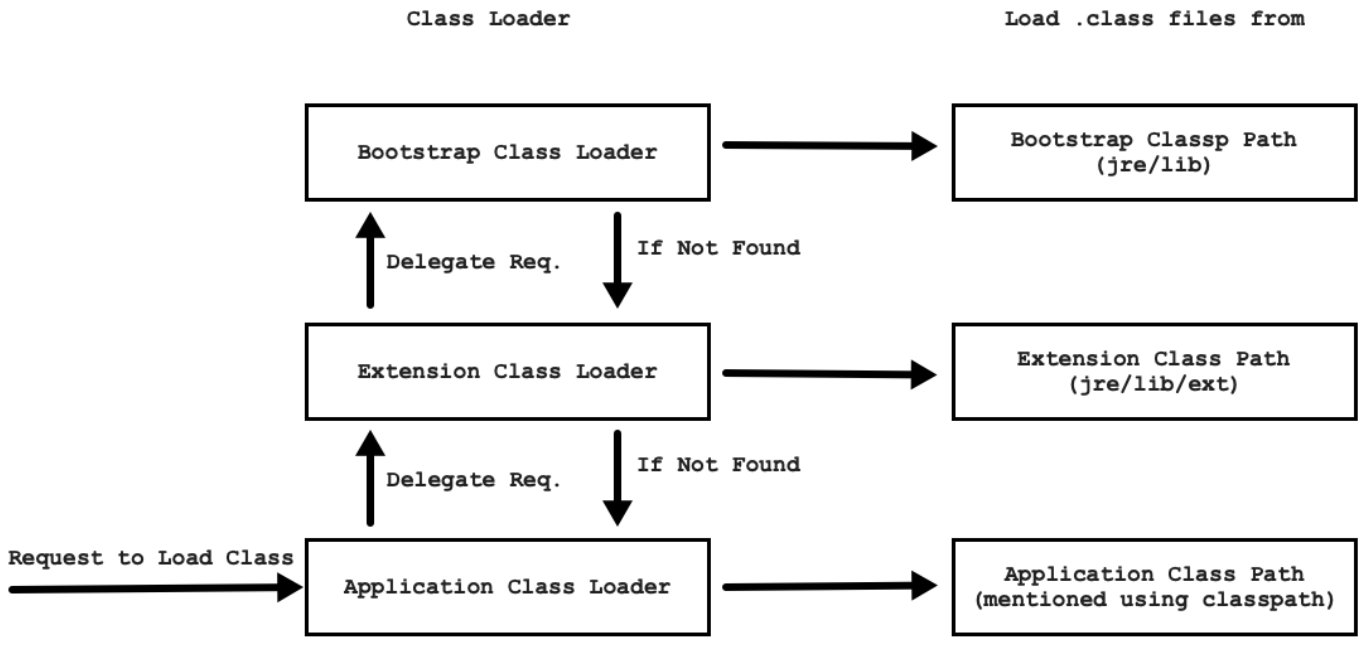
- Image Reference: https://gist.github.com/bossiernesto/ccb3a847e83ae0ddf7db0b0eae30870f

Class Loader Subsystem



- Class loader subsystem is responsible for loading Java classes dynamically into Java Virtual Machine. It loads, links and initializes the class file when it refers to a class for the first time at runtime, not compile time.

- **Loading**
  - Bootstrap class loader, Extension class loader and Application / System class loader helps to load classes.
  - The Class Loader Hierarchy in Java works using a **delegation model**, where each class loader delegates the class loading responsibility to its parent before attempting to load a class itself.



- **Bootstrap class loader:**
  - It is the parent of all other class loaders and is typically implemented in native code as part of the JVM implementation.
  - jre/lib path is called as bootstrap classpath.
  - Bootstrap Class Loader loads core Java classes from the bootstrap classpath, which includes the classes from the rt.jar file.
  - Consider below code:

```
class Program{
  public static void main( String[] args )throws Exception{
          Class<?> c = String.class;
          ClassLoader classLoader = c.getClassLoader();
          if( classLoader == null )
                  System.out.println("Bootstrap ClassLoader");
          else
                  System.out.println("Either Extension/Application ClassLoader");
  }
}
```

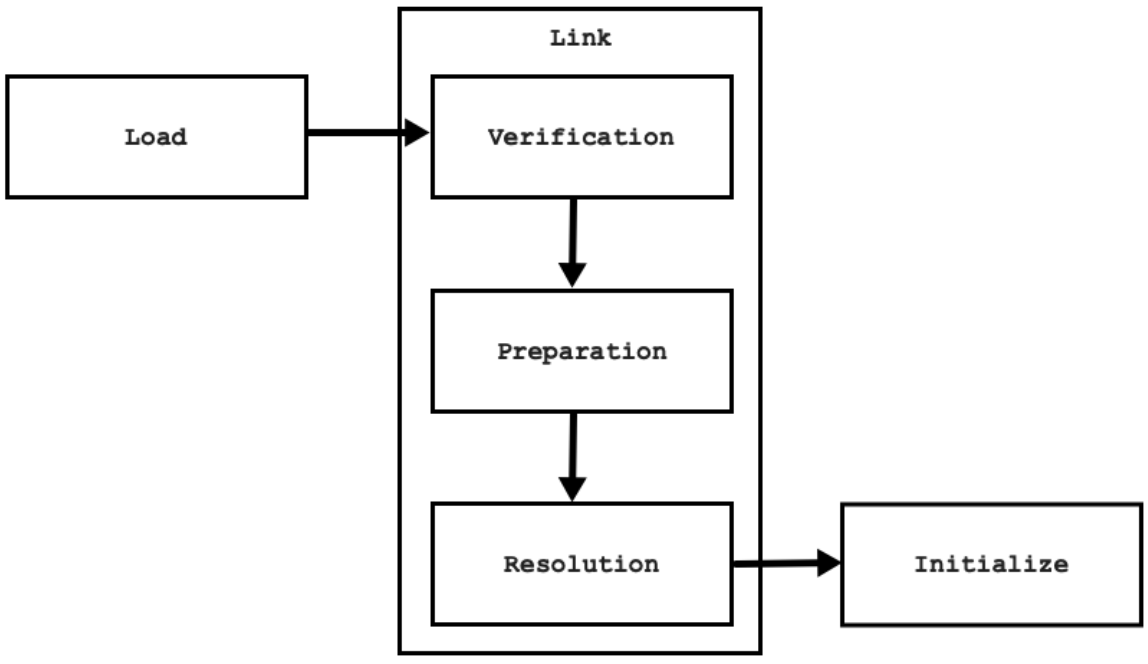
- **Extension class loader:**
  - It is a child of the bootstrap class loader and a parent of the application class loader.
  - It loads classes from the extension directories(jre/lib/ext).
  - In Java 8, extension directories are directories where we can place additional libraries or extensions that we want to be loaded by the Extension ClassLoader. These extensions can provide extra functionality to our Java applications without modifying the core Java installation.
  - How to check extension class loader programatically?
    - Copy MySQL connector into jre/lib/ext directory.

```
class Program{
  public static void main( String[] args ){
    ClassLoader classLoader = Class.forName("com.mysql.cj.jdbc.driver").getClassLoader();
    System.out.println( classLoader );
  }
}
```

- **Application class loader**
  - It is a child of the extension class loader and the last class loader in the hierarchy.
  - It loads classes from the application classpath, which includes user-defined classes and libraries.
  - Consider below code:

```
class Program{
  public static void main( String[] args ){
    ClassLoader classLoader = Program.class.getClassLoader();
    System.out.println( classLoader );
  }
}
```

- **Linking:**
  - It consists of three distinct steps: verification, preparation, and resolution.

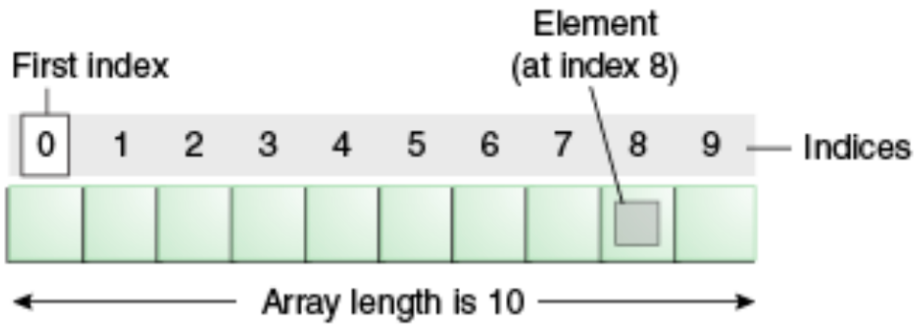


- **Verification:**
  - Bytecode verifier is responsible for doing verification.
  - In this step, the JVM verifies the binary format of the class file(.class) to ensure it complies with the JVM specifications and does not violate any security constraints.
  - This verification process checks for various aspects, such as the correctness of the class file structure, proper use of bytecode instructions, and adherence to access control rules.
- **Preparation:**
  - In the preparation step, the JVM allocates memory for class variables and initializes them with default values. This includes static fields and other static data structures associated with the class.
  - The memory allocation and initialization are done at this stage to ensure that the class is ready for use when it is loaded into memory.
- **Resolution:**
  - In Java code, when we refer to a class, method, or field using its name, we are using a symbolic reference. For example, calling a method printRecord() on an object reference emp is a symbolic reference to the method print( ).
  - Direct references, on the other hand, are the actual memory addresses or offsets that point to the location of the class, method, or field in memory. For example, a direct reference to the print method would be the memory address where the bytecode instructions for that method are located.
  - Resolution is the process of linking symbolic references in the class to concrete references. This involves translating symbolic references to direct references that can be used during runtime.
  - Resolution includes three tasks:
    - **Class Resolution:** Resolving references to other classes used by the current class, ensuring that they exist and can be accessed.
    - **Interface Resolution:** Resolving references to interfaces that the class implements, verifying that the interfaces are valid and can be used.
    - **Field and Method Resolution:** Resolving references to fields and methods used by the class, validating their existence and accessibility.

- **Initialization:**
  - It is a final setp in classloader subsystem of the Java Virtual Machine.
  - It involves executing the static initialization blocks and initializing static variables defined in the class.

Array

- Data structure and algorithm are two different branches of computer science. Data structure is all about efficient memory organization whereas algorithm is about processing data stored in data structure.
- Elements that are grouped together is called as collection. In short data structure group elements together.
  - Linear Data structures
    - Array
    - Stack
    - Queue
    - LinkedList
  - Non Linear Data structures
    - Tree
    - Graph
    - Hashtable
- Array is a linear/sequential data structure/collection which is used to store multiple elements of same type in continuous memory location. Value or instance stored in data structure is called as element.



An array of 10 elements.

  - To access elements of array, we should use integer index.
  - Array index always begins with 0. Hence last index of element will be array size -1.
  - Array size is always fixed. It means that we can not grow or shrink array dynamically.
- Array is a non primitive type in Java. It means that array instance will get space on heap.
- Types of Array:
  - Single dimensional array
  - Multi dimensional array

- Ragged array

Single Dimensional Array

- **How to declare reference for single dimensional array?**

```
int arr[ ]; //OK
int [arr];  //Not OK
int[ ] arr; //OK
```

- **How to create array instance?**
  - Option 1:

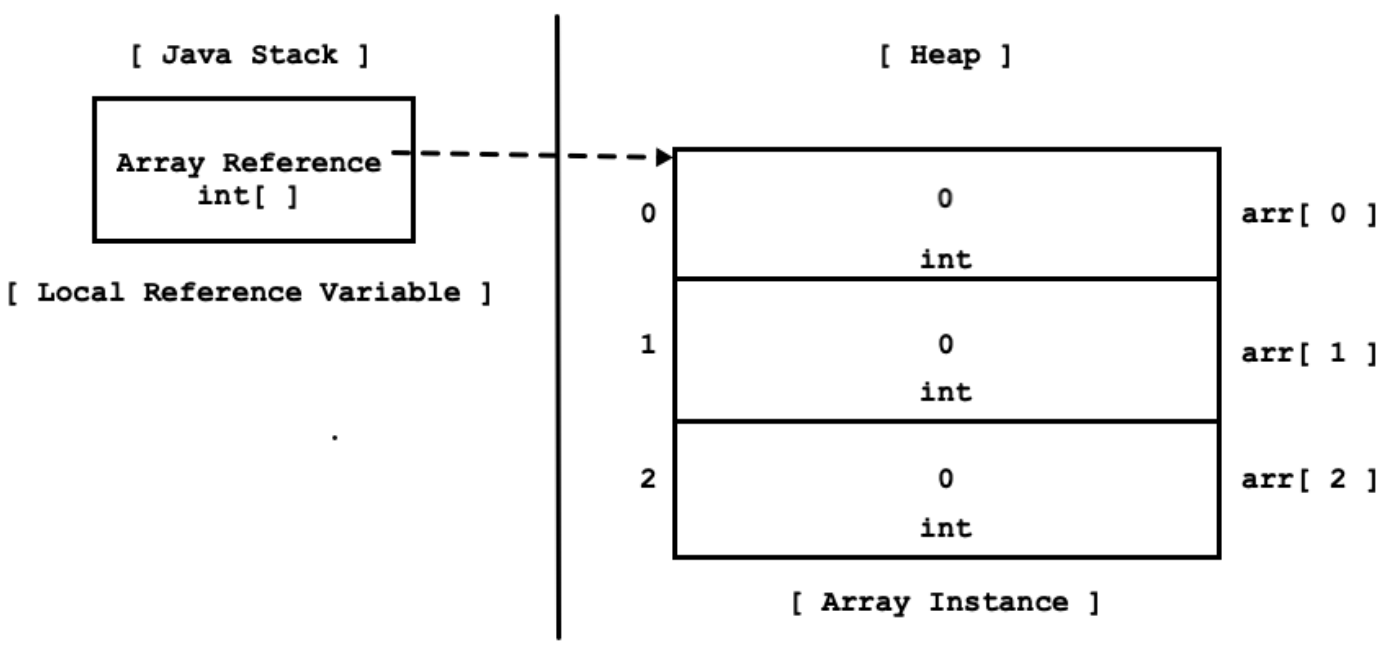
```
int[] arr;
arr = new int[ 3 ]; //OK
```

  - Option 2:

```
int[] arr = new int[ 3 ]; //OK
```

  - Option 3:

```
System.out.print("Enter size  : ");
int size = sc.nextInt( ); //Assuming size 3
int[] arr = new int[ size ];  //OK
```



- What will happen if size is negative?

```
int[] arr = new int[ -3 ];    //Not OK: NegativeArraySizeException
```

- If we try to create an array with negative size then JVM throw NegativeArraySizeException.
- **How to print elements of Array:**
  - Option 1: Using do while loop

```
int[] arr = new int[ 3 ];
int index = 0;
do{
  System.out.println( arr[ index ] );
  ++ index;
}while( index < 3 );
```

  - Option 2: Using while loop

```
int[] arr = new int[ 3 ];
int index = 0;
while( index < 3 ){
  System.out.println( arr[ index ] );
  ++ index;
}
```

- Option 3: Using for loop

```
int[] arr = new int[ 3 ];
for( int index = 0; index < 3; ++ index ){
  System.out.println( arr[ index ] );
}
```

- Option 4: Using foreach loop

```
int[] arr = new int[ 3 ];
for( int element : arr ){  //for each element in arr
  System.out.println( element );
}
```

  - foreach loop is also called as iterator.
  - It is used only to read value and only in forward direction.
  - We will discuss it again in Iterable and Iterator.
- Option 5: Using java.util.Arrays.toString() method

```
int[] arr = new int[ 3 ];
System.out.println( arr );  //[I@6d06d69c
String str = Arrays.toString( arr );
System.out.println( str );  //[0, 0, 0]
```

- Consider example: [I@6d06d69c
  - [represents depth nesting.
  - @ is a character
  - 6d06d69c is a hexadecimal hashcode

```
double[] arr = new double[ 3 ];
System.out.println( arr );    //[D@6d06d69c
```

```
double[][] arr = new double[ 3 ][ 3 ];
System.out.println( arr );    //[[D@6d06d69c
```

```
double[][][] arr = new double[ 3 ][ 3 ][ 3];
System.out.println( arr );    //[[[D@6d06d69c
```

- Option 6: Using Java 8 Streams

```
int[] arr = new int[ 3 ];
Arrays.stream(arr).forEach(System.out::println);
//Method Reference: System.out::println
```

- **Element Type and its encoding:**

Element Type	Encoding
boolean	Z
byte	B
char	C
class or interface	L*classname;*
double	D
float	F
int	I
long	J
short	S

- **How to initialize array?**

```
int[] arr = new int[ 3 ]; //OK
```

```
int[] arr = new int[ 3 ]{ 10, 20, 30 }; //Not OK
```

```
int[] arr = new int[  ]{ 10, 20, 30 }; //OK
```

```
int[] arr = { 10, 20, 30 }; //OK
```

- **ArrayIndexOutOfBoundsException**

```
public static void main(String[] args) {
  int[] arr = new int[ ] { 10, 20, 30 };

  //System.out.println( arr[ -1 ] );  //ArrayIndexOutOfBoundsException

  System.out.println( arr[ 0 ] ); //10
  System.out.println( arr[ 1 ] ); //20
  System.out.println( arr[ 2 ] ); //30

  //System.out.println( arr[ 3 ] );   //ArrayIndexOutOfBoundsException
}
```

  - Using illegal index, if we try to access elements of array then JVM throws ArrayIndexOutOfBoundsException.

- **How to pass array as a argument to the another method?**

```
private static Scanner sc = new Scanner( System.in );
public static void acceptRecord( int[] arr ){
  if( arr != null ){
    for( int index = 0; index < arr.length; ++ index ){
      System.out.print("Enter element : ");
      arr[ index ] = sc.nextInt( );
    }
  }
}
public static void printRecord( int[] arr ){
  if( arr != null ){
    for( int index = 0; index < arr.length; ++ index ){
      System.out.print(arr[ index+" "] );
    }
    System.out.println( );
  }
}
public static void main( String[] args ){
  int[] arr = new int[ 3 ];
  Program.acceptRecord( arr );
  Program.printRecord( arr );
}
```

- In Java, .length is a property of arrays that represents the number of elements in the array.

- **ArrayStoreException**
  - Example 1:

```
StringBuilder[] arr = new StringBuilder[ 3 ]; //OK
arr = new String[ 3 ];  //Not OK
```

  - Example 2:

```
String[] arr = new String[ 3 ];  //OK
arr = new StringBuilder[ 3 ]; //Not OK
```

  - Example 3:

```
Object[] arr = new String[ 3 ];  //OK
arr = new StringBuilder[ 3 ]; //OK
```

  - Example 4:

```
Object[] arr = new String[ 3 ];  //OK
arr[ 0 ] = "Sandeep Kulange";                //OK
arr[ 1 ] = new String("Ahmednagar");  //OK
arr[ 2 ] = String.valueOf( 414101 );  //OK
```

  - Example 5:

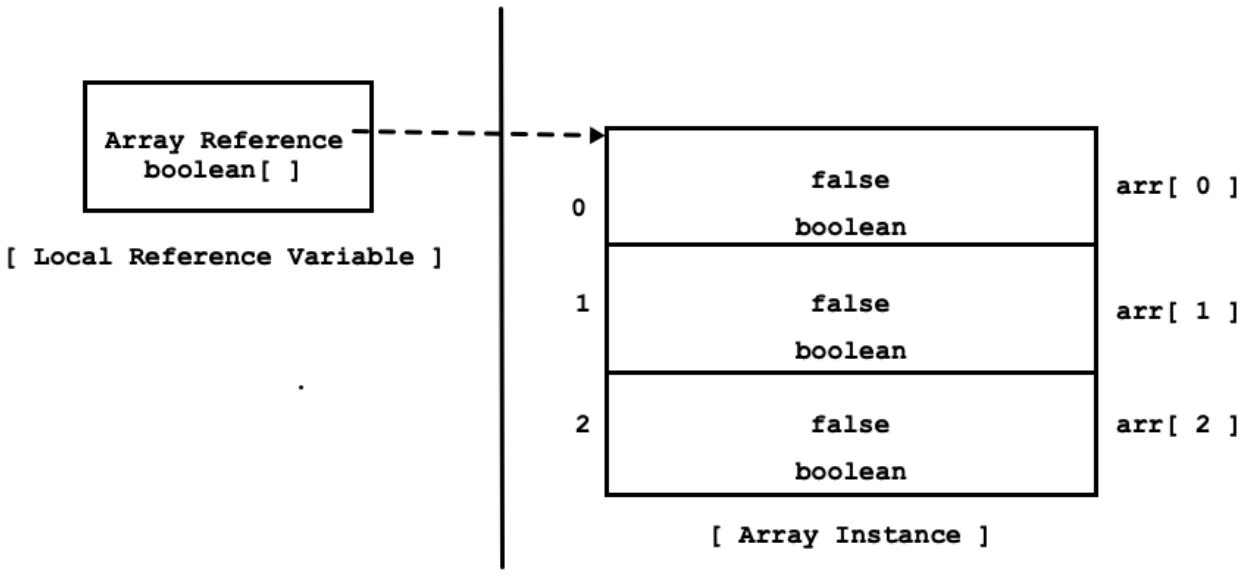
```
Object[] arr = new String[ 3 ];  //OK
arr[ 0 ] = "Sandeep Kulange";                //OK
arr[ 1 ] = new String("Ahmednagar");  //OK
arr[ 2 ] = new StringBuilder( "414101" );  //Not OK: ArrayStoreException
```

    - When we try to store wrong type of instance into array then JVM throws ArrayStoreException.

- **Array of value type versus Array of references versus Array of instances**
  - **Array of primitive type / value type**
    - If we create array of primitive type / value type then it contain value.
    - Default value in array depends on default value of array type.

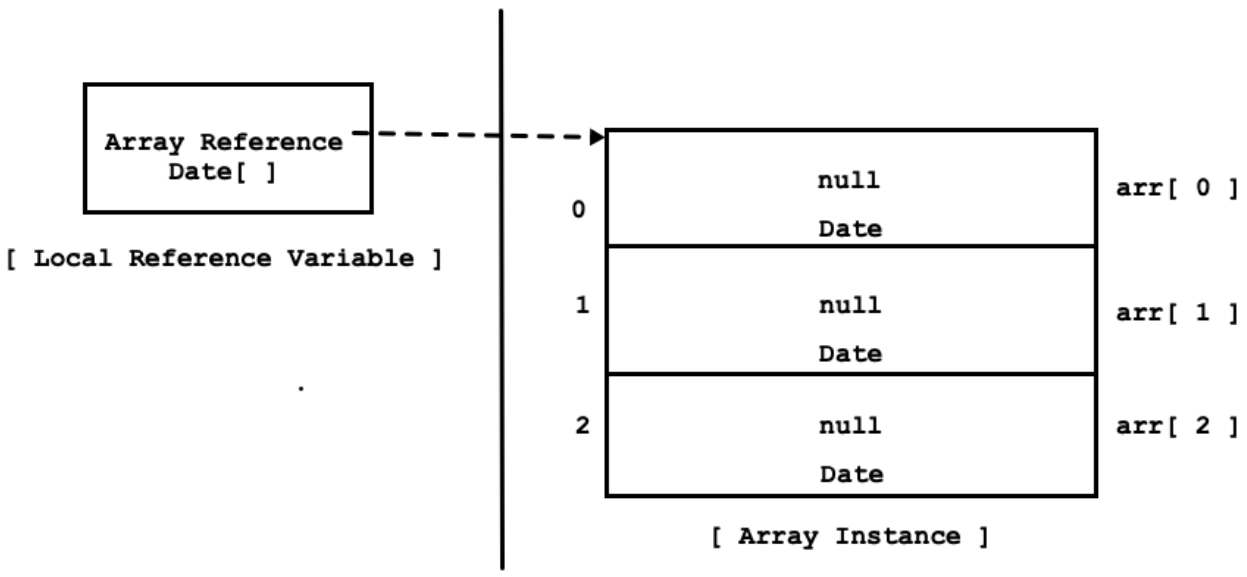
- In below code, type of array is boolean and default value of boolean is false.

```
boolean[] arr = new boolean[ 3 ];
```



- **Array of references**
  - If we create array of references then it can contain reference not a value.
  - Reference value can be reference of instance of non primitive type or null.
  - In below code "Date[] arr" is reference of array and "new Date[3]" is Array instance in which default value of all the elements will be null.

```
Date[] arr = new Date[ 3 ];
```

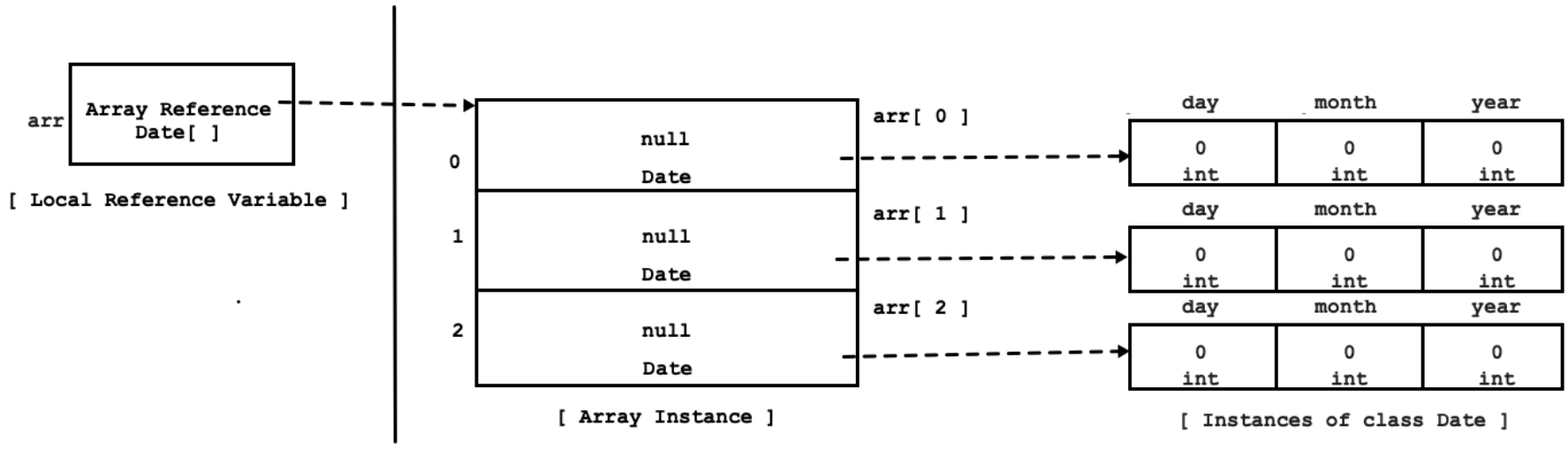


- **Array of instances of reference type / non primitve type**
  - Option 1:

```
Date[ ] arr = new Date[ 3 ];
arr[ 0 ] = new Date( );
arr[ 1 ] = new Date( );
arr[ 2 ] = new Date( );
```

  - Option 2:

```
Date[ ] arr = new Date[ 3 ];
for( int index = 0; index < arr.length; ++ index )
  arr[ index ] = new Date( );
```



- **Passing argument to the method by value verus by reference.**
  - In Java, we can not pass argument to the method by reference or by address. Any variable(primitive/non primitive) is passed to the method by value only.
  - Example 1:

```
public class Program {
  public static void swap( int x, int y ) {
    int temp = x;
    x = y;
    y  = temp;
  }
  public static void main(String[] args) {
    int a = 10;
    int b = 20;

    Program.swap(a, b);   //a and b are passed to method by value

    System.out.println("a :   "+a);   //10
    System.out.println("b :   "+b);   //20
  }
}
```

  - Example 2:

```
class Date{
  private int day, month, year;
  public Date(int day, int month, int year) {
    this.day = day;
    this.month = month;
    this.year = year;
  }
  public String toString() {
    return this.day+" / "+this.month+" / "+this.year;
  }
}
public class Program {
  public static void swap( Date dt1, Date dt2) {
    Date temp = dt1;
    dt1 = dt2;
    dt2 = temp;
  }
  public static void main(String[] args) {
    Date dt1 = new Date(23, 7, 1983);
    Date dt2 = new Date(21, 9, 2021);

    Program.swap(dt1, dt2);   //dt1 and dt2 are passed to the method by value

    System.out.println(dt1.toString());   //23 / 7 / 1983
    System.out.println(dt2.toString());   //21 / 9 / 2021
  }
}
```

  - Using array we can simulate pass by reference concept in Java. But remember, we can not pass argument to the method by reference.
    - Example 3:

```
public class Program {
  public static void swap( int[] arr) {
    int temp = arr[ 0 ];
    arr[ 0 ] = arr[ 1 ];
    arr[ 1 ] = temp;
  }
  public static void main(String[] args) {
    int a = 10;
    int b = 20;

    int[] arr = new int[ ] { a, b };
    Program.swap( arr );  //arr is passing to the method by value
    a = arr[ 0 ];
    b = arr[ 1 ];

    System.out.println(a);    //20
    System.out.println(b);    //10
  }
}
```

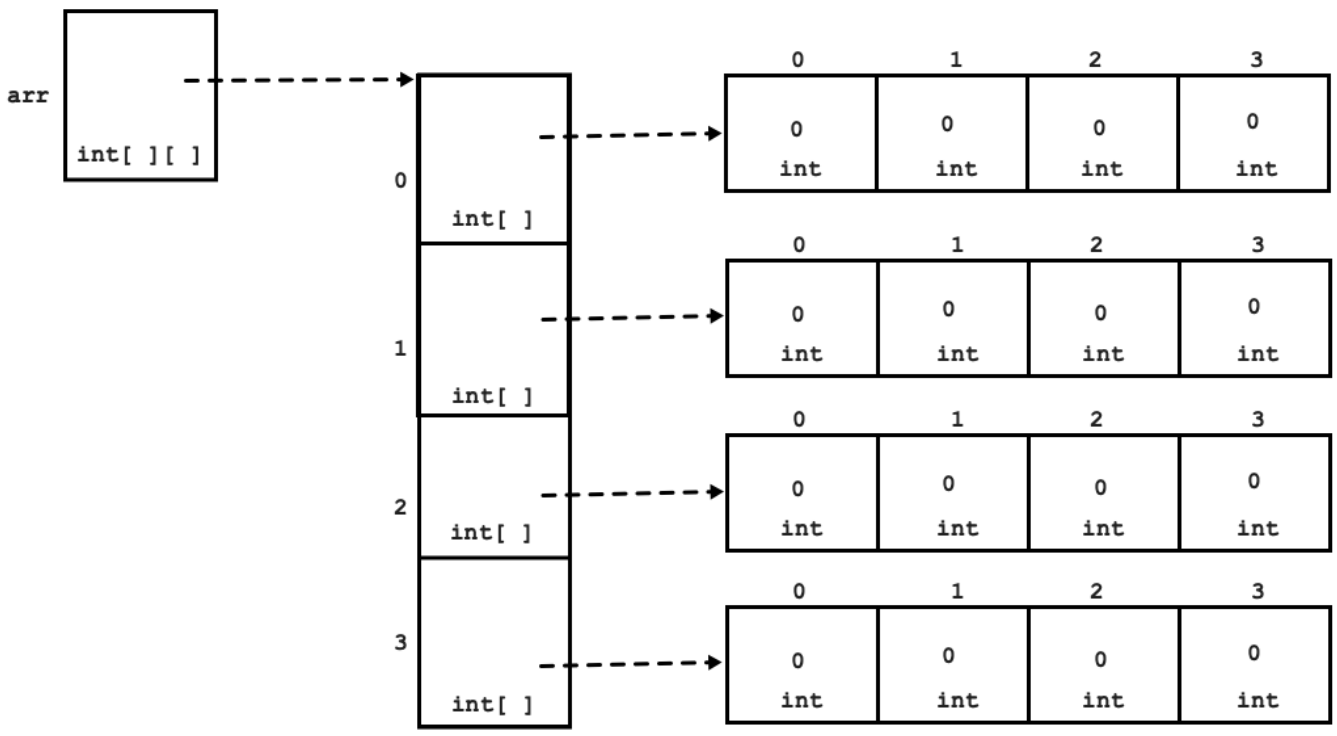
Multi Dimensional Array

- An array of array which is having fixed size is called as multidimensional array.
  - In multi dimensional array element is array
  - length of every element is same.
- **How to declare reference for multi dimensional array?**

```
int arr[][];  //OK
int[] arr[];  //OK
int[][] arr;  //OK
```

- **How to create instance of multi dimensional array?**

```
int[][] arr = new int[ 4 ][ 4 ];  //OK
```



- **How to initialize multi dimensional array?**
  - Option 1:

```
int[][] arr = new int[][]{
    {1,2,3,4},
    {5,6,7,8},
    {9,10,11,12},
    {13,14,15,16}
  };
```

- Option 2:

```
int[][] arr = {
    {1,2,3,4},
    {5,6,7,8},
    {9,10,11,12},
    {13,14,15,16}
  };
```

- **How to print elements of multi dimensional array?**
- Option 1: Using do-while loop

```
int[][] arr = { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 }, { 13, 14, 15, 16 } };
int row = 0;
  do {
      int col = 0;
      do {
          System.out.print(arr[row][col] + " ");
          col = col + 1;
      } while (col < arr[row].length);
      System.out.println();
      row = row + 1;
  } while (row < arr.length);
```

- Option 2: Using while loop

```
int[][] arr = { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 }, { 13, 14, 15, 16 } };
int row = 0;
while (row < arr.length) {
  int col = 0;
  while (col < arr[row].length) {
    System.out.print(arr[row][col] + " ");
    col = col + 1;
  }
  System.out.println();
  row = row + 1;
}
```

- Option 3: Using for loop

```
int[][] arr = { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 }, { 13, 14, 15, 16 } };

  for( int row = 0; row < arr.length; ++ row ) {
      for( int col = 0; col < arr[ row ].length; ++ col ) {
          System.out.print(arr[ row ][ col ]+"    ");
      }
      System.out.println();
  }
```

- Option 4: Using foreach loop

```
int[][] arr = { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 }, { 13, 14, 15, 16 } };

  for( int[] row : arr ) {
      for( int col : row ) {
          System.out.print(col+"  ");
      }
      System.out.println();
  }
```

- Option 5: Using Java 8 Stream API

```
int[][] arr = { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 }, { 13, 14, 15, 16 } };

  Arrays.stream(arr)
```

```
    .flatMapToInt(Arrays::stream)
    .forEach(System.out::println
```

- Option 6: Using toString and deepToString method

```
int[][] arr = { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 }, { 13, 14, 15, 16 } };

  System.out.println( arr );  //[[I@6d06d69c

  for( int row = 0; row < arr.length; ++ row ) {
      System.out.println( Arrays.toString(arr[ row ]) );
  }
```

```
int[][] arr = { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 }, { 13, 14, 15, 16 } };
  System.out.println( Arrays.deepToString(arr) );
```

- **How to pass multidimensional array as a argument to the method**

```
public class Program {
  private static void acceptRecord(int[][] arr) {
    //TODO: Logic for accepting array
  }
  private static void printRecord(int[][] arr) {
    //TODO: Logic for print array
  }
  public static void main(String[] args) {
    int[][] arr = new int[ 3 ][ 3 ];
    Program.acceptRecord( arr );
    Program.printRecord( arr );
  }
}
```

- Multi-dimensional arrays can be used in various situations where data needs to be organized in more than one dimension, such as representing images, storing tabular data.

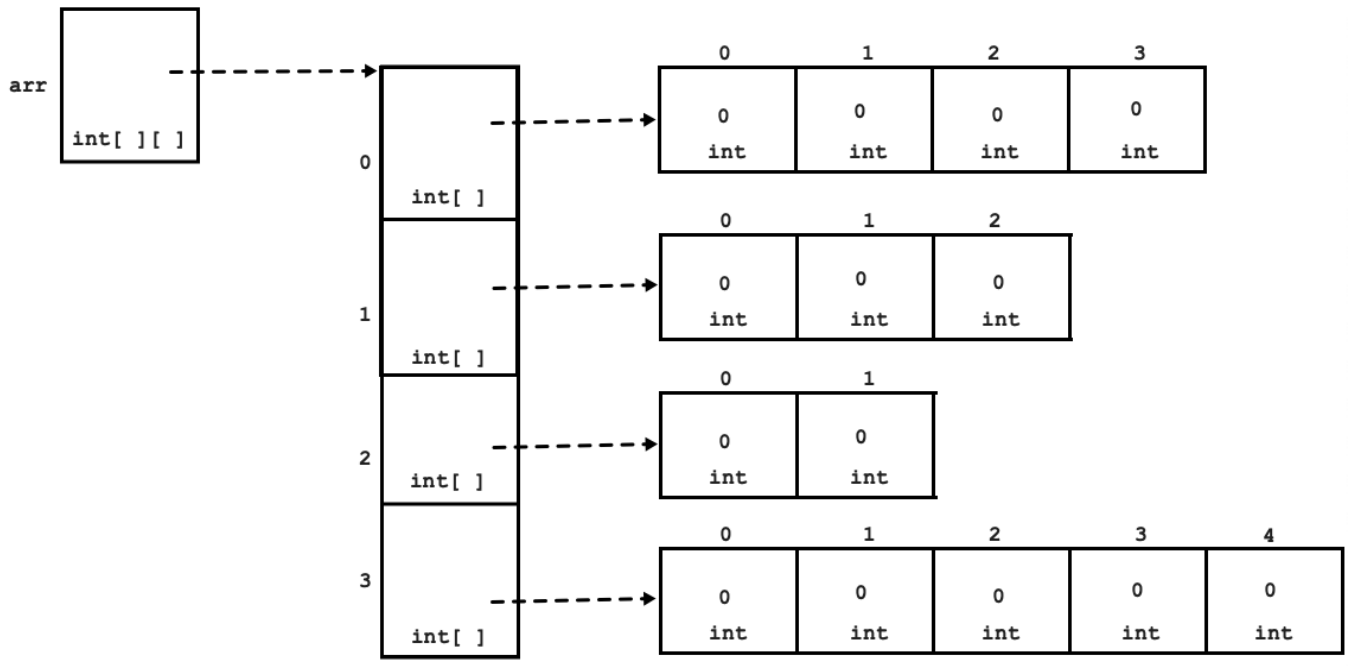
Ragged Array

- An array of array in which array size is different is called as ragged array.
  - In ragged array element is array
  - length of every element is different.
- **How to declare reference for ragged array?**

```
int arr[][];  //OK
int[] arr[];  //OK
int[][] arr;  //OK
```

- **How to create instance of ragged array?**

```
int[][] arr = new int[ 4 ][  ];   //Array of 4 references
arr[ 0 ] = new int[ 4 ];          //Single dimensional array of 4 elements
arr[ 1 ] = new int[ 3 ];          //Single dimensional array of 3 elements
arr[ 2 ] = new int[ 2 ];         //Single dimensional array of 2 elements
arr[ 3 ] = new int[ 4 ];         //Single dimensional array of 4 elements
```



- **How to initialize ragged array?**

```
int[][] arr = new int[ 4 ][  ];
arr[ 0 ] = new int[  ]{ 1, 2, 3, 4 };
arr[ 1 ] = new int[  ]{ 5, 6, 7} ;
arr[ 2 ] = new int[  ]{ 8, 9 };
arr[ 3 ] = new int[  ]{ 10, 11, 12, 13 };
```

```
int[][] arr = {
  { 1, 2, 3, 4 },
  { 5, 6, 7 },
  { 8, 9},
  { 10, 11, 12, 13 }
};
```

- **How to print ragged array?**
- Option 1: Using do-while loop

```
int[][] arr = { { 1, 2, 3, 4 }, { 5, 6, 7 }, { 8, 9 }, { 10, 11, 12, 13 } };
int row = 0;
  do {
      int col = 0;
      do {
          System.out.print(arr[row][col] + " ");
          col = col + 1;
      } while (col < arr[row].length);
      System.out.println();
      row = row + 1;
  } while (row < arr.length);
```

- Option 2: Using while loop

```
int[][] arr = { { 1, 2, 3, 4 }, { 5, 6, 7 }, { 8, 9 }, { 10, 11, 12, 13 } };
int row = 0;
while (row < arr.length) {
  int col = 0;
  while (col < arr[row].length) {
    System.out.print(arr[row][col] + " ");
    col = col + 1;
  }
  System.out.println();
  row = row + 1;
}
```

- Option 3: Using for loop

```
int[][] arr = { { 1, 2, 3, 4 }, { 5, 6, 7 }, { 8, 9 }, { 10, 11, 12, 13 } };
  for( int row = 0; row < arr.length; ++ row ) {
      for( int col = 0; col < arr[ row ].length; ++ col ) {
          System.out.print(arr[ row ][ col ]+"    ");
      }
      System.out.println();
  }
```

- Option 4: Using foreach loop

```
int[][] arr = { { 1, 2, 3, 4 }, { 5, 6, 7 }, { 8, 9 }, { 10, 11, 12, 13 } };
  for( int[] row : arr ) {
      for( int col : row ) {
          System.out.print(col+"  ");
      }
      System.out.println();
  }
```

- Option 5: Using Java 8 Stream API

```
int[][] arr = { { 1, 2, 3, 4 }, { 5, 6, 7 }, { 8, 9 }, { 10, 11, 12, 13 } };

  Arrays.stream(arr)
  .flatMapToInt(Arrays::stream)
  .forEach(System.out::println
```

- Option 6: Using toString and deepToString method

```
int[][] arr = { { 1, 2, 3, 4 }, { 5, 6, 7 }, { 8, 9 }, { 10, 11, 12, 13 } };

  System.out.println( arr );  //[[I@6d06d69c

  for( int row = 0; row < arr.length; ++ row ) {
      System.out.println( Arrays.toString(arr[ row ]) );
  }
```

```
int[][] arr = { { 1, 2, 3, 4 }, { 5, 6, 7 }, { 8, 9 }, { 10, 11, 12, 13 } };
  System.out.println( Arrays.deepToString(arr) );
```

- **How to pass ragged array as a argument to the method**

```
public class Program {
  private static int[][] getArray( ){
    int[][] arr = new int[ 4 ][  ];
    arr[ 0 ] = new int[ 4 ];
    arr[ 1 ] = new int[ 2 ];
    arr[ 2 ] = new int[ 3 ];
    arr[ 3 ] = new int[ 4 ];
    return arr;
  }
  private static void acceptRecord(int[][] arr) {
    //TODO: Logic for accepting array
  }
  private static void printRecord(int[][] arr) {
    //TODO: Logic for print array
  }
  public static void main(String[] args) {
    int[][] arr = Program.getArray( );
    Program.acceptRecord( arr );
    Program.printRecord( arr );
  }
}
```

- **References:**

- https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html
- https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html
- https://commons.apache.org/proper/commons-lang/javadocs/api-release/org/apache/commons/lang3/ArrayUtils.html

Variable arity parameters(varargs)

- Consider overloaded sum method:

```
class Program{
  public static int sum( ){
    return 0;
  }
  public static int sum( int number ){
    return number;
  }
  public static int sum( int num1, int num2 ){
    return num1 + num2;
  }
  public static int sum( int num1, int num2, int num3, int num4, int num5 ){
    return num1 + num2 + num3 + num4 + num5;
  }
  public static void main( String[] args ){
    Program.sum( );
    Program.sum( 10 );
    Program.sum( 10, 20 );
    Program.sum( 10, 20, 30, 40, 50);
  }
}
```

- If overloaded method accepts parameters of same type then we can use Java language feature i.e variable arity parameter.
- It allows us to pass a variable number of arguments of the same type to a method. Consider below code:

```
class Program{
  public static int... numbers ){
    int result = 0;
    for( int number : numbers )
      result = result + number;
    return result;
  }

  public static void main( String[] args ){
    Program.sum( );
    Program.sum( 10 );
    Program.sum( 10, 20 );
    Program.sum( 10, 20, 30, 40, 50);
  }
}
```

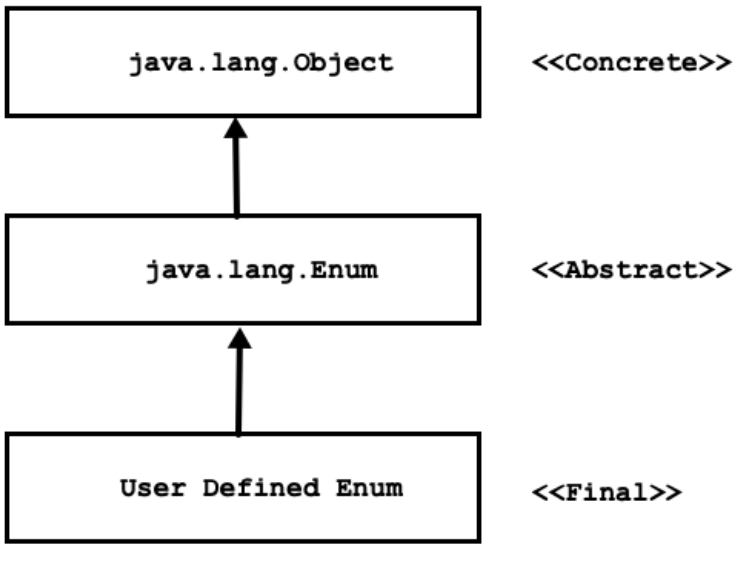
- Points to remember about varargs in Java:
  - The ellipsis (...) must be the last parameter in the method's parameter list.
  - We can only have one varargs parameter per method.
  - Inside the method, the varargs parameter behaves like an array, allowing us to iterate over its elements using a loop or access individual elements by index.
  - If the method is called with no arguments, the varargs parameter will be an empty array.
  - If the method is called with multiple arguments, those arguments will be converted into an array and passed to the method.
  - We can specify variable arity parameter for constructor as well as main method.

Enum

- In programming languages like C, C++, Java etc, If we want to improve readability of the source code then we should give proper name to the identifier i.e type/field/method/variable/parameter etc.
- But what about the costants or literals? Can we give name to the constant or literals? Answer to this question is Yes.
- In C/C++, Java, Using enum, we can give name to the constant/literals.
  - For C/C++, refer https://en.cppreference.com/w/c/language/enum
- As we know, there are 4 non primitive (or reference) types:
  - Interface

- Class
- Enum
- Array

- enum is non primitive type in Java. Every enum we define in the code is implict final class which is sub class of java.lang.Enum class. Consider below hierarchy:



  - Below are the methods of java.lang.Object class
    - public String toString();
    - public boolean equals(Object obj);
    - public native int hashCode();
    - protected native Object clone() throws CloneNotSupportedException;
    - protected void finalize() throws Throwable;
    - public final native Class<?> getClass();
    - public final void wait() throws InterruptedException;
    - public final void wait(long timeout) throws InterruptedException;
    - public final void wait(long timeout, int nanos) throws InterruptedException;
    - public final native void notify();
    - public final native void notifyAll();
  - Below are the methods of java.lang.Enum class
    - public final String name();
    - public final int ordinal();
    - public final Class getDeclaringClass();
    - public static <T extends Enum> T valueOf(Class, String);
    - Also contains method from Object class and Comparable interface.

- Let us first understand basics of enum and then we will learn how to give name to the constant/literal.
- Define Language enumeration using enum keyword:

```
//Language.java
public enum Language {
    MARATHI,
    HINDI,
    ENGLISH,
    FRENCH,
    GERMAN
}
```

  - In above example, name of the enum is Language and MARATHI, HINDI, ENGLISH, FRENCH, GERMAN are enum constants / enumerators.
    - Name of first enumerator in MARATHI and its ordinal value is 0.
    - Name of second enumerator in HINDI and its ordinal value is 1.
    - Name of third enumerator in ENGLISH and its ordinal value is 2.
    - Name of fourth enumerator in FRENCH and its ordinal value is 3.
    - Name of fifth enumerator in GERMAN and its ordinal value is 4.
  - **Remember** ordinal value begins with 0. Once we declare enumerator, we can not change its ordinal value.

- Let us decompile enum using javap.
  - https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javap.html

```
javac Language.java
javap -c Language.class
```

- Below is converted final class from enum.

```
final class Language extends Enum<Language> {
  public static final Language MARATHI; //Reference of Language enum
  public static final Language HINDI;   //Reference of Language enum
  public static final Language ENGLISH; //Reference of Language enum
  public static final Language FRENCH;  //Reference of Language enum
  public static final Language GERMAN;  //Reference of Language enum

  public static Language[] values(); //Returns array of enumerators
  public static Language valueOf(String name);
}
```

- How to get name and ordinal of MARATHI enumerator.
  - Option 1:

```
public static void main( String[] args ){
  String name = Language.MARATHI.name();
  int ordinal = Language.MARATHI.ordinal();
  System.out.println("Enumerator Name : "+name)
  System.out.println("Enumerator Ordinal : "+ordinal)
}
```

  - Option 2:

```
public static void main( String[] args ){
  Language language = Language.MARATHI;
  String name = language.name();
  int ordinal = language.ordinal();
  System.out.println("Enumerator Name : "+name)
  System.out.println("Enumerator Ordinal : "+ordinal)
}
```

- How to get name and ordinal of all enumerator.
  - Option 1:

```
public static void main( String[] args ){
  Language language = Language.MARATHI;
  System.out.printf("%-15s%-5d\n", language.name(), language.ordinal());

  language = Language.HINDI;
  System.out.printf("%-15s%-5d\n", language.name(), language.ordinal());

  language = Language.ENGLISH;
  System.out.printf("%-15s%-5d\n", language.name(), language.ordinal());

  language = Language.FRENCH;
  System.out.printf("%-15s%-5d\n", language.name(), language.ordinal());

  language = Language.GERMAN;
  System.out.printf("%-15s%-5d\n", language.name(), language.ordinal());
}
```

  - Option 2:

```
public static void main( String[] args ){
  Language[] languages = Language.values( );
  for( Language language : languages ){
    System.out.printf("%-15s%-5d\n", language.name(), language.ordinal());
  }
}
```

- How to get reference of enum using enumerator?

```
  public static void main( String[]  args ){
    String name = "FRENCH";
    Language language = Language.valueOf( name );
    System.out.printf("%-15s%-5d\n", language.name(), language.ordinal());
  }
```

```
public static void main( String[]  args ){
  String name = "ITALIAN";
  Language language = Language.valueOf( name ); //IllegalArgumentException
  System.out.printf("%-15s%-5d\n", language.name(), language.ordinal());
}
```

- If pass illegal or inappropriate argument to the method then JVM throws IllegalArgumentException
- Now Let us see, how to give name to single litreal using enum.

```
  enum Language {
    MARATHI("MR"), HINDI("HI"), ENGLISH("EN"), FRENCH("FR"), GERMAN("DE");
  }
```

- If we want to give name to the literal then we should define constructor inside enum. Consider below code:

```
  enum Language {
    MARATHI("MR"), HINDI("HI"), ENGLISH("EN"), FRENCH("FR"), GERMAN("DE");  //It must be first line

    String codeName;
    Language(String codeName) {
      this.codeName = codeName;
    }
  }
```

  - **Remember** since enums are designed to represent a fixed set of constants, explicit instantiation is restricted. Hence access modifier of enum constructor should be either private or package private.

- If you take example of MARATHI enumerator then name() will return MARATHI and ordinal will return 0. The how to get value of codeName?

```
  enum Language {
    MARATHI("MR"), HINDI("HI"), ENGLISH("EN"), FRENCH("FR"), GERMAN("DE");  //It must be first line in enum
definition

    String codeName;
    Language(String codeName) {
      this.codeName = codeName;
    }

    public String getCodeName( ){
      return this.codeName;
    }
  }
```

```
  public static void main( String[] args ){
    Language language = Language.HINDI;
    System.out.println("Name  : "+language.name());             //HINDI
    System.out.println("Ordinal  : "+language.ordinal);         //1
    System.out.println("CodeName  : "+language.getCodeName());  //HI
  }
```

- Now Let us see, how to give name to multiple litreals using enum.

```
  enum Country {
    USA("United States", "US"), INDIA("India", "IN"), UK("United Kingdom", "GB");

    private final String countryName;
    private final String countryCode;
```

```
  Country(String countryName, String countryCode) {
    this.countryName = countryName;
    this.countryCode = countryCode;
  }

  public String getCountryName( ){
    return this.countryName;
  }

  public String getCountryCode( ){
    return this.countryCode;
  }
}
```

```
public static void main( String[] args ){
  Country country = Country.INDIA;
  System.out.println("Name  : "+country.name());                    //INDIA
  System.out.println("Ordinal  : "+country.ordinal);                //1
  System.out.println("Country Name  : "+country.getCountryName());  //India
  System.out.println("Country Code  : "+country.getCountryCode());  //IN
}
```

- We can overload constructor in enum. Consider below code:

```
public enum Day {
  SUNDAY("Sun", 1), MONDAY("MON"), TUESDAY(3), WEDNESDAY("Wed", 4);

  private String name;
  private int number;

  Day(String name){ //Overloaded constructor
    this.name = name;
  }
  Day(int number){ //Overloaded constructor
    this.number = number;
  }
  Day( String name, int number ){ //Overloaded constructor
    this.name = name;
    this.number = number;
  }

  public String getName( ){
    return this.name;
  }
  public String getNumber( ){
    return this.number;
  }
  //We can override methods too
  public String toString( ){
    return String.format("%-15s%-5d", this.name, this.number);
  }
}
```

- We will discuss inheritance and interface in upcoming session. But remember, enum class can not extend any other class explicitly(implicitly extends java.lang.Enum) but it can implement interface(s). Consider below code:

```
interface Printable{
  void print( );
}
enum Color implements Printable{
  RED, GREEN, BLUE;

  public void print( ){
      System.out.printf("%-10s%-5d\n", this.name(), this.ordinal());
  }
}
class Program{
  public static void main( String[] args ){
    Color color = Color.GREEN;
    color.print( );
```

```
  }
}
```

- How to use enum to write menu driven code?

```
enum Shape {
  EXIT, RECTANGLE, CIRCLE, TRIANGLE;
}
```

```
private static Scanner sc = new Scanner( System.in );
public static Shape menuList( ){
  System.out.println("0. Exit");
  System.out.println("1. Rectangle");
  System.out.println("2. Circle");
  System.out.println("3. Triangle");
  System.out.prin("Enter choice:  ");
  return Shape.values( )[ sc.nextInt()];
  //Shape.values( ) returns array from it returning enumerator at sc.next( ) index.
}
```

```
public static void main( String[] args ){
  Shape choice;
  while( ( choice = Program.menuList( ) ) != Shape.EXIT ){
    switch( choice ){
      case RECTANGLE:
        //TODO: method calls
        break;
      case CIRCLE:
        //TODO: method calls
        break;
      case TRIANGLE:
        //TODO: method calls
        break;
    }
  }
}
```

- Reference: https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html