JAVA

Lecture X – Generic

ARRAYLIST

- What is the problem of array?
- Cannot change its size
- We are asked to pick up all the numbers that is greater than 10, but we don't know how many integers are there.
- Use java. util.ArrayList, can be seen as dynamic array
- ArrayList provides a wide range of useful methods to manipulate a collection of elements.

```
ArrayList<Type> name = new ArrayList<>();
```

ARRAYLIST

- Useful ArrayList methods:

```
boolean add(E e); // add an element to the tail
void add(int index, E e); // add an element at a specified position
void clear();// remove all elements
boolean contains(Object o); // check if it contains a specified element
E get(index i);// get an element at a specified position
E remove(index i);// remove an element at a specified position
boolean remove(Object o);// remove the first occurrence of an object
int size();// return the size of the arraylist
```

https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#ArrayList--

EXAMPLE:ARRAYLIST

```
class ArrayListDemo{
      public static void main(String[] args){
            ArrayList<Character> a1 = new ArrayList<>();
            a1.add('A');
            a1.add('B');
            a1.add(1, 'C');
            a1.remove(2);
            …
      }
}
```

TYPE CONVERSION

- Can we read integer and floating numbers from the command-line?

```
java Args 1 2
```

- We want 1 and 2 as integer number!

- Automatic type conversion:
  - Two types are compatible.
  - The destination type is larger than the source type.
  - E.g., byte to int, int to long, long to double, …
- Cast: an instruction to the compiler to convert one type into another

```
(target-type) expression
```

TYPE CONVERSION

```
double x, y;
// …
int z = (int) (x / y);
```

- Narrowing conversion: information might be lost.
- E.g., information lost when we convert long to short

- Example: CastDemo

- How to convert string into integer, double, …

TYPE WRAPPER

- Type Wrapper: classes that encapsulate the primitive types.
- Primitive types: are not objects, e.g., cannot be passed by reference.
- Double, Integer, Float, …

- Numeric wrappers provide methods to convert a string into corresponding number.
- Double.parseDouble(String)
- Integer.parseInt(String)
- Short.parseShort(String)
- …
- Boolean values:
- Boolean.parseBoolean(String)

ARRAYLIST

- Useful ArrayList methods:

```
boolean add(E e); // add an element to the tail
void add(int index, E e); // add an element at a specified position
void clear();// remove all elements
boolean contains(Object o); // check if it contains a specified element
E get(index i);// get an element at a specified position
E remove(index i);// remove an element at a specified position
boolean remove(Object o);// remove the first occurrence of an object
int size();// return the size of the arraylist
```

https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#ArrayList--

GENERICS

- Which methods are different from what we haven seen before?

GENERICS

- Which methods are different from what we haven seen before?
- `boolean add(E e) // what is type E?`
- `E get(int index) // what is returned?`
- Are ArrayList<Integer> and ArrayList<Character> the same?

GENERICS

- Which methods are different from what we haven seen before?
- `boolean add(E e) // what is type E?`
- `E get(int index) // what is returned?`
- Are ArrayList<Integer> and ArrayList<Character> the same?
- NO, they are different types.
- Generics: the capability to parameterize types.
  - E is a generic type, i.e., e is an object of the generic type
  - Automatically work with the type of data passed to its type parameter.

EXAMPLE: GENERICS

```
class Gen<T>{ // T is a generic type
      T ob; // declare a reference to an object of type T
      Gen(T o){ // constructor
            ob = o;
      }
      T getOb(){
            return ob;
      }
      void showType(){
            System.out.println(ob.getClass().getName());
      }
}
```

OBJECT VS GENERICS

- Object is the ancestor of all classes, i.e., all classes can be seen as an object .
- Can we use Object to generalise classes, interface and methods?
- `Object method(Object o)`

OBJECT VS GENERICS

- Object is the ancestor of all classes, i.e., all classes can be seen as an object .
- Can we use Object to generalise classes, interface and methods?
- `Object method(Object o)`

- What are the problems?
- Object cannot be used to safely convert to its actual type.
- e.g., `Object method(Object o)` always returns an Object but not its actual type.
- Unless o is casted to its actual type, it cannot call its own methods.

GENERICS

- General form:

```
class class-name<type-param-list>{
        body…
}
```

- Properties on Generics:
  - Works only with Object type, e.g., why ArrayList<int> fails?
  - Different versions of a generic type are not type-compatible, e.g., ArryList<Integer> and ArrayList<Double>
  - We can use more than one generic types, e.g,. Gen<V, T>

EXAMPLE

```
class NumType<T>{
	T num;
	NumType(T t){
		num = t;
	}
	double multiply(double x){
		return num.doubleValue() * x;
	}
}
```

BOUNDED TYPES

```
class NumType<T extends Number>{
	T num;
	NumType(T t){
		num = t;
	}
	double multiply(double x){
		return num.doubleValue() * x;
	}
}
```

BOUNDED TYPES

```
class NumType<T extends Number>{
        T num;
        NumType(T t){
                num = t;
        }
        boolean absEquals(NumType<T> ob){
                return Math.abs(num.doubleValue()) ==
Math.abs(ob.num.doubleValue());
        }
}
```

What is the problem?

WILDCARD ARGUMENTS

```
class NumType<T extends Number>{
        T num;
        NumType(T t){
                num = t;
        }
        boolean absEquals(NumType<?> ob){
                return Math.abs(num.doubleValue()) ==
Math.abs(ob.num.doubleValue());
        }
}
```

What is the problem?

GENERIC METHODS

It is possible to have generic method defined in a non-generic class

```
class GenericMethod{
	static <T, V> boolean arrayEquals(T[] x, V[] y){
		…
	}
}
```

GENERIC CONSTRUCTOR

Also it is possible to have generic constructor

```
class Summation{
	private int sum;
	<T extends Number> Summation(T arg){
		for(int i = 0; i < arg.intValue(); i++){
			sum += i;
		}
	}
}
```

STRING

- One of the most important data structure in Java.
  - Strings are objects in Java, not primitive type.

- Construct a String:
- `String str = new String("Happy"); // like an object`
- `String str2 = new String(str); // from another string`

- Alternatively
- `String str = "Happy";`

STRING METHODS

- Useful methods that operate on String:
- `boolean equals(Object str)//` return true, if they contains the same character sequence
- `int length() //` return the number of characters
- `char charAt(int index) //` return character at a specified index
- `int compareTo(String str) //` comparison based on Unicode of each character
- `int indexOf(char ch/ string str) //` return the index of the first occurrence of the given character or substring
- `int lastIndexOf(char ch/ string str) //` last index of..
- `String[] split(String regex) //` split string by given regular expression .e.g. "\\s+", splits the input string based on one or more whitespace characters.

- You can find more here
- https://docs.oracle.com/javase/8/docs/api/java/lang/String.html

IMMUTABLE STRING

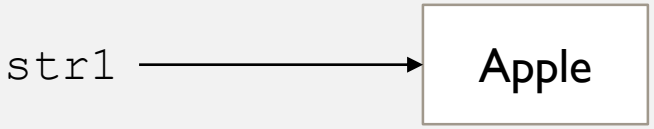
- The contents of a String object are immutable.
- Example, replace returns a string resulting from replacing all occurences of oldChar in this string with newChar

```
String replace(char oldChar, char newChar)

String str1 = “Apple”;

str1.replace(’p’, ‘b’); // will it change the value of str1?

String str2 = str1.replace(’p’, ‘b’);
```

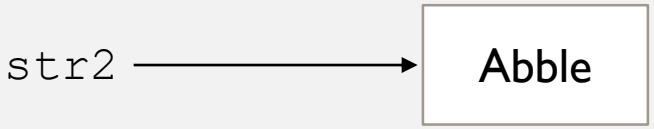
IMMUTABLE STRING

- The contents of a String object are immutable.
- Example, replace returns a string resulting from replacing all occurences of oldChar in this string with newChar

```
String replace(char oldChar, char newChar)
String str1 = "Apple";
str1.replace('p', 'b');
String str2 = str1.replace('p', 'b');
```



replace



Once String "Apple" is created, it cannot be changed.

```
str1 = str1.replace('p', 'b');
// update the reference, "Apple" Garbage Collected
```

What if we need to manipulate a string in several steps? StringBuffer, StringBuilder

STRING AND CHAR[]

- Both String and char[] represent a collection of characters, are they the same?

```
String str1 = "Happy";
for(char c : str1){
      ...
} // Will it work?

char[] cs1 = {'a', 'b', 'c'};
char[] cs2 = {'d', 'e', 'f'};
char[] cs3 = cs1 + cs2; // Will it work?
```

STRING VS CHAR[]

- Data type: Single data type vs collections
- Immutable: Immutable vs mutable
- Build-in functions: String has a lot of build-in functions, char[] not.
- Accessing each character: charAt() vs var_name[index]
- Conversions:

```
String s = "Happy";
char[] cs = s.toCharArray();
cs = {'H', 'A', 'P', 'P', 'Y'};
s = new String(cs);
```

COURSEWORK 1

- Dfile.encoding=UTF-8: This sets the file encoding to UTF-8, which specifies the character encoding for the input and output files.
- XX:+UseSerialGC: This flag instructs the Java Virtual Machine (JVM) to use the Serial Garbage Collector for memory management.
- Xss64m: This sets the thread stack size to 64 megabytes. This is the maximum stack size that each thread of the Java application can use.
- Xms1920m: This sets the initial heap size to 1920 megabytes. This is the initial amount of memory allocated to the Java application when it starts.
- Xmx1920m: This sets the maximum heap size to 1920 megabytes. This is the maximum amount of memory that the Java application can use.
- PGP < input.txt > Output.txt is the actual command to run PGP. It reads input from a file named input.txt and writes the output to a file.

KEYWORDS

```
abstract continue for new switch assert default goto package synchronized
boolean do if private this break double implements protected throw byte
else import public throws case enum instanceof return transient catch
extends int short try char final interface static void class finally long
strictfp volatile const float native super while
```

- Learning what each of these means takes you a long way in learning the language
- https://www.geeksforgeeks.org/list-of-all-java-keywords/

VARIABLE NAMING CONVENTION

Type	Rules	Example
Class	• Start with uppercase letter • Noun	`class Employee{` `}`
Interface	• Start with uppercase letter • Adjective	`interface Runnable{` `}`
Method	• Start with lowercase letter • Verb • CamelCase, if multiple words	`void draw{` `}`
Variable	• Start with lowercase letter • CamelCase, if multiple words	`int id;`
Constant	• In uppercase • Multiple words should be separated by _ • May contain digits	`MIN_AGE = 18;`

NAMING CLASSES

- **Specific** names encourage small, cohesive classes
- Good class name are **Nouns**
- **Single** Responsibility avoid **Generic** Names (Manager->FileManager, DataBaseManager, or ResourceManager…)

KEYWORDS

```
abstract continue for new switch assert default goto package synchronized
boolean do if private this break double implements protected throw byte
else import public throws case enum instanceof return transient catch
extends int short try char final interface static void class finally long
strictfp volatile const float native super while
```

- Learning what each of these means takes you a long way in learning the language
- https://www.geeksforgeeks.org/list-of-all-java-keywords/

NAMING METHOD

- With a good method name, the reader doesn't need to read the method to know what is does
- **Avoid Side Effects**: make sure the method name telling the truth; **Naming Warning Signs**: And, If, Or
- Avoid **Abbreviations**

NAMING VARIABLES:BOOLEANS

- Don't: open, start, status, login;
- Do: isOpen, done, isActive, loggedIn
- E.g. if(login){} to if(loggedIn){}

MAGIC NUMBERS

- If(age>18){}?
- const int lEGAL_DRINKING_AGE = 18;
- if(age> lEGAL_DRINKING_AGE){
  - //
  - }

BE POSITIVE

- Don't: if(!isNotLoggedIn)
- Do: if(loggedIn)

POLYMORPHISM IN PRACTICE

```
Public void LoginUser(User user){
      switch(user.Status){
            case Status.Active:
            //active user logic
            break;
            case Status.Inactive;
            // inactive user logic
            break;
            case Status.Locaked;
            // locked user logic
            break;
```

USER CLASS

```
public abstract class User{
        // attibutes
        public abstract void login();
}

public class ActiveUser extends User{
        public void login(){
                //Active user logic here
…
```

WHAT ELSE

- Design pattern
- Java Docs
- Github java app source code
- IDE
- Unit test
- LeetCode