Functions

Learning Outcomes:

After successful completion of this lesson, you should be able to:

- Identify the different string and character functions used in Java;
- Discuss the different mathematical function;
- Explain purpose of using global and local variables

In Java, all function definitions must be inside classes. We also call functions methods. Let's look at an example method

```
public class Main {
    public static void foo() {
        // Do something here
    }
}
```

foo is a method we defined in class Main. Notice a few things about foo.

- static means this method belongs to the class Main and not to a specific instance of Main. Which means we can call the method from a different class like that Main.foo().
- void means this method doesn't return a value. Methods can return a single value in Java and it has to be defined in the method declaration. However, you can use return by itself to exit the method.
- This method doesn't get any arguments, but of course Java methods can get arguments as we'll see further on.

Arguments

Arguments to Java methods are passed by value, although some might disagree with my choice of words, I find it the best way to explain and understand how it works exactly.

By value means that arguments are copied when the method runs. Let's look at an example.

```
public void bar(int num1, int num2) {
                ...
}
```

Here is a another place in the code, where bar is called

```
int a = 3;
int b = 5;
bar(a, b);
```

You can picture in your head that when bar(a, b) is run, it's like in the beginning of bar the following two lines are written:

```
int num1 = a;
int num2 = b;
```

And only then the rest of the method is run.

This means that a value get copied to num1 and b value get copied to num2. Changing the values of num1 and num2 will not affect a and b.

If the arguments were objects, the rules remain the same, but it acts a bit differently. Here is an example:

```
public void bar2(Student s1, Student s2) {
    ...
}
```

And here is how we use it

```
Student joe = new Student("joe");
Student jack = new Student("jack");
bar2(joe, jack);
```

Again we can picture the same two lines in the beginning of bar2:

```
Student s1 = joe;
Student s2 = jack;
```

But when we assign objects, it's a bit different than assigning primitives. s1 and joe are two different references to the same object. s1 == joe is true. This means that running methods on s1 will change the object joe. But if we'll change the value of s1 as a reference, it will not affect the reference joe.

```
s1.setName("Chuck"); // joe name is now
Chuck as well
s1 = new Student("Norris"); // s1 is a new
student, different than joe with the name of
Norris
// s1 == joe   is not true anymore
```

Non static methods

Non static methods in Java are used more than static methods. Those methods can only be run on objects and not on the whole class.

Non static methods can access and alter the field of the object.

```
public class Student {
    private String name;
    public String getName() {
        return name;   }
    public void setName(String name) {
        this.name = name;
    } }
```

Calling the methods will require an object of type Student.

```
Student s = new Student();
s.setName("Danielle");
String name = s.getName();

Student.setName("Bob"); // Will not work!
Student.getName(); // Will not work!
```

Summary

- Every Java method has to be within a class
- Static methods belong to a class while non-static methods belong to objects
- All parameters to functions are passed by value, primitives content is copied, while objects are not copied and some would say 'passed by reference'

Java Predefined Functional Interface

- The java.util.function package defines several predefined functional interfaces that you can use when creating lambda expressions or method references.
- They are widely used throughout the Java API

Functional Interface	Abstract Method	Function descriptor	Description
Consumer<T>	accept(T t)	T -> void	Represents an operation that accepts a single input argument and returns no result.
Function<T, R>	apply(T t)	T -> R	Represents a function that accepts one argument and produces a result.
Predicate<T>	test(T t)	T -> boolean	Represents a predicate (boolean-valued function) of one argument.
Supplier<T>	get()	() -> T	Represents a supplier of results.

Example using Consumer<T>

```
public class ConsumerApp {
  public static void main(String[] args) {
    String[] players = {"Rafael Nadal", "Novak Djokovic",
      "Stanislas Wawrinka", "David Ferrer",
      "Roger Federer", "Andy Murray", "Tomas Berdych",
      "Juan Martin Del Potro", "Richard Gasquet", "John Isner"};
    // Show the list of players
    System.out.print("Show the list of players:\n");
    // void forEach(Consumer<? super T> action)
    Arrays.asList(players).forEach((player) -> System.out.println(player)); } }
```

Output:

```
Show the list of players:
Rafael Nadal
Novak Djokovic
Stanislas Wawrinka
David Ferrer
Roger Federer
Andy Murray
Tomas Berdych
Juan Martin Del Potro
Richard Gasquet
John Isner
```

Example using Function<T , R>

```java
public class FunctionApp {
  public static void main(String[] args) {
    String[] players = {"Rafael Nadal", "Novak Djokovic",
      "Stanislas Wawrinka", "David Ferrer",
      "Roger Federer", "Andy Murray",
      "Tomas Berdych", "Juan Martin Del Potro",
      "Richard Gasquet", "John Isner"};
    Function<String[],String> converter = (all)-> {
      String names= "";
      for (String n : all){
        String forname=n.substring(0, n.indexOf(" "));
        forname=n.substring(n.indexOf(" "))+" "+forname;
        names+=forname+"\n";
      }
      return names;
    };
    System.out.println(converter.apply(players));
  }
}
```

Output:

```
Nadal Rafael
 Djokovic Novak
 Wawrinka Stanislas
 Ferrer David
 Federer Roger
 Murray Andy
 Berdych Tomas
 Martin Del Potro Juan
 Gasquet Richard
 Isner John
```

Example using Predicate<T>

```
public class PredicateApp {
  private static List getBeginWith(List<String> list, Predicate<String> valid) {
    List<String> selected = new ArrayList<>();
    list.forEach(player -> {
      if (valid.test(player)) {
        selected.add(player);
      }
    });
    return selected;
  }

  public static void main(String[] args) {
    String[] players = {"Rafael Nadal", "Novak Djokovic",
      "Stanislas Wawrinka", "David Ferrer",
      "Roger Federer", "Andy Murray", "Tomas Berdych",
      "Juan Martin Del Potro", "Richard Gasquet", "John Isner"};
    List playerList = Arrays.asList(players);
    System.out.println(getBeginWith(playerList,(s)->s.startsWith("R")));
    System.out.println(getBeginWith(playerList,(s)->s.contains("D")));
    System.out.println(getBeginWith(playerList,(s)->s.endsWith("er")));
  }
}
```

Output:

```
[Rafael Nadal, Roger Federer, Richard Gasquet]
[Novak Djokovic, David Ferrer, Juan Martin Del Potro]
[David Ferrer, Roger Federer, John Isner]
```

Example using Supplier<T>

```
public class SupplierApp {
  private static void printNames(Supplier<String> arg) {
    System.out.println(arg.get());
  }
  private static void listBeginWith(List<String> list, Predicate<String> valid) {
    printNames(()->"\nList of players:");
    list.forEach(player -> {
      if (valid.test(player)) {
        printNames(()->player);
      }
    });
  }
  public static void main(String[] args) {
    String[] players = {"Rafael Nadal", "Novak Djokovic",
      "Stanislas Wawrinka", "David Ferrer",
      "Roger Federer", "Andy Murray", "Tomas Berdych",
      "Juan Martin Del Potro", "Richard Gasquet", "John Isner"};
    List playerList = Arrays.asList(players);
    // print which starts with 'R'
    listBeginWith(playerList, (s) -> s.startsWith("R"));
    listBeginWith(playerList, (s) -> s.contains("D"));
    listBeginWith(playerList, (s) -> s.endsWith("er"));
  }
}
```

Output:

```
List of players:
Rafael Nadal
Roger Federer
Richard Gasquet

List of players:
Novak Djokovic
David Ferrer
Juan Martin Del Potro

List of players:
David Ferrer
Roger Federer
John Isner
```

String Functions

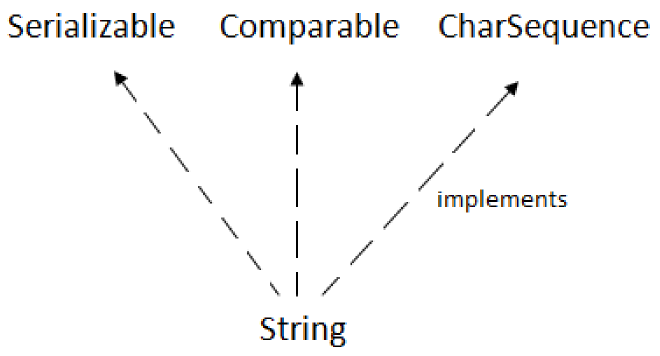
In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string. For example:

```
char[] ch={'j','a','v','a','t','p','o','i','n','t'};
String s=new String(ch);
```

Is same as:

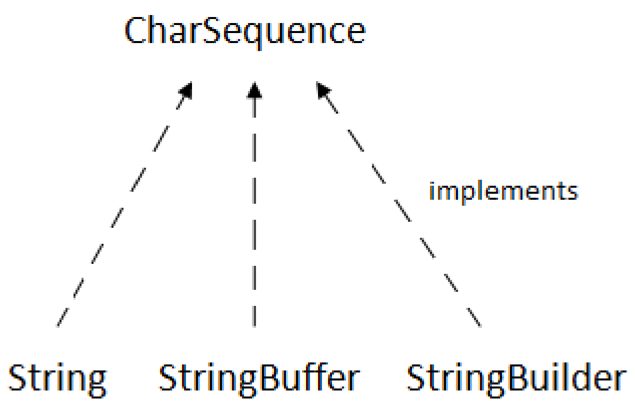
```
String s="javatpoint";
```

Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() et



CharSequence Interface

The CharSequence interface is used to represent the sequence of characters. String, StringBuffer and StringBuilder classes implement it. It means, we can create strings in java by using these three classes.



The Java String is immutable which means it cannot be changed. Whenever we change any string, a new instance is created. For mutable strings, you can use StringBuffer and StringBuilder classes.

What is string in Java?

Generally, String is a sequence of characters. But in Java, string is an object that represents a sequence of characters. The java.lang.String class is used to create a string object.

There are two ways to create String object:

- By string literal
- By new keyword

Java String literal is created by using double quotes. For Example:

```
String s="welcome";
```

Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

```
String s1="Welcome";
String s2="Welcome";//It doesn't create a new instance
```

By new keyword In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).

```
String s=new String("Welcome");//creates two objects and one reference variable
```

Java string example:

```
public class StringExample{
public static void main(String args[]){
String s1="java";//creating string by java string literal
char ch[]={'s','t','r','i','n','g','s'};
String s2=new String(ch);//converting char array to string
String s3=new String("example");//creating java string by new keyword
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
}}
```