Lambdas : Functional Programming

==>Reduce the boiler-plate code for simple activity

==>activity to behave dynamically

```
 greetTeam(<behavior>){
      behavior();
 }
```

funVar : shall hold def. not the value returned by function

```
 int a=10;
 String str="Hello";
funVar=public void show(){
                -----
                ------
         }

greetTeam(<action>){
}

greetTeam(funVar){
}
```

```
FunctionInstance funVar=public void show(){
                    -----
                    ------
             }
```

#not provided any specific type

# Use interface to represent the function type

```
interface GreetingBehavior{
      void show();
}
```

```
GreetingBehavior funVar=public void show(){
                    -----
                    ------
             }
```

function def assigned to instance of interface will be actually an implementation of show method that is a part of interface

```
GreetingBehavior funVar=() -> {
                    -----
                    ------
             }
```

Lambdas can be implemented for those interface only , having only single abstract method

==>Functional-Interface

class

anonymous inner

enum

```
funVar=()-> System.out.println();
funVar=(a, b)-> a+b;  //no param type req..
funVar=(a, b)-> a+b;   //if single stmt, not bounded in braces , by default it is associated with
return

funVar=(a,b)->{
     return a+b;
}
funVar=a-> System.out.println(a);  //single param,no brackets
```

Lambda expression can match with any functional interface method as long as prototype matches...

Lambda Ex ~ Interfaces

#Backward Compatibility

Collection (students...)

=>sort records based on some criteria

=>filter

#enclosing elements shall be final (pre-jdk 8)