

Function

- L'interfaccia funzionale `Function<T, R>` definisce un metodo chiamato `apply()` che prende in ingresso un oggetto di tipo generico `T` e ritorna un tipo generico `R`

```
Function<String, Integer> stringLength = x -> x.length();
```

- In questo esempio, il metodo `apply()` ha in ingresso un parametro `String`, l'implementazione è `x.length()`, e restituisce il valore dato da `length()`
- Nel frammento seguente, `map()` usa la funzione definita sopra `stringLength` per calcolare la lunghezza di ogni parola dello stream, e quindi `reduce()` per sommare le lunghezze

```
int result =  
    Stream.of("truth", "flows", "to", "them", "sweetly", "by", "nature")  
        .map(stringLength)  
        .reduce(0, Integer::sum);
```

```
System.out.println(result);  
// Output: 31
```

1

Prof. Tramontana - Maggio 2020

Supplier

- Un `Supplier<T>` è un'interfaccia funzionale che ha un singolo metodo chiamato `get()` e rappresenta una funzione che non prende in ingresso alcun parametro e restituisce un valore di tipo `T`

```
Supplier<String> sup = () -> "ciao ciao";
```

```
String s = sup.get();
```

```
// s: ciao ciao
```

- Il codice sopra implementa un `Supplier` che restituisce una stringa, quando è chiamato il metodo `get()`. Il supplier non tiene una stringa ma il codice per generarla

2

Prof. Tramontana - Maggio 2020

Factory Method (Breve Ripasso)

- Il design pattern Factory Method permette di creare oggetti e di nascondere la loro classe al client che li usa

```
public class Creator {  
    public static Prodotto getProdotto(String name) {  
        switch (name) {  
            case "primo": return new ProdottoA();  
            case "secondo": return new ProdottoB();  
            case "terzo": return new ProdottoC();  
            case "quarto": return new ProdottoD();  
            default: return new ProdottoA();  
        }  
    }  
}
```

- Nel metodo della classe `Creator` si sceglie quale classe istanziare
- `ProdottoA`, `ProdottoB` e `ProdottoC` sono sottotipi di `Prodotto`
- Tipicamente si avrà un chiamante

```
Prodotto p = Creator.getProdotto("primo");
```

3

Prof. Tramontana - Maggio 2020

Factory Method Con Supplier

- Usando un `Supplier`
`Supplier<Prodotto> prodSupplier = ProdottoA::new;`
- La linea di codice sopra è equivalente a
`Supplier<Prodotto> suppl = () -> new ProdottoA();`
- Per avere istanze di sottotipi di `Prodotto`, si chiama `get()` sul `Supplier`
`Prodotto p1 = prodSupplier.get();`
- Quindi si crea una mappa che fa corrispondere al nome di un `Prodotto` la sua creazione
`Map<String, Supplier<Prodotto>> map = Map.of("primo", ProdottoA::new, "secondo", ProdottoB::new, "terzo", ProdottoC::new);`
- Si usa la mappa per istanziare sottotipi di `Prodotto`

```
public static Prodotto getProdotto(String name) {  
    Supplier<Prodotto> s = map.get(name);  
    if (s != null)  
        return s.get();  
    return new ProdottoA();  
}
```
- Il frammento di codice sopra è equivalente a

```
public static Prodotto getProdotto(String name) {  
    return map.getOrDefault(name, ProdottoA::new).get();  
}
```
- Se un costruttore ha parametri in ingresso si dovrà usare un'interfaccia funzionale appropriata (diversa da `Supplier`)

4

Prof. Tramontana - Maggio 2020

Classe Persona

```
public class Persona {
    private String nome, ruolo;
    private int eta, costo;

    public Persona(String n, int e, String r, int c) {
        nome = n;
        eta = e;
        ruolo = r;
        costo = c;
    }

    public int getCosto() {
        return costo;
    }

    public int getEta() {
        return eta;
    }

    public String getNome() {
        return nome;
    }

    public String getRuolo() {
        return ruolo;
    }
}
```

5

Prof. Tramontana - Maggio 2020

Operazioni Su Istanze Di Persona

- Data una lista di istanze di Persona, stampare e contare i nomi dei programmatori

```
private List<Persona> team = List.of(
    new Persona("Al", 28, "Architect", 44),
    new Persona("Claire", 29, "Programmer", 38),
    new Persona("Ed", 26, "Programmer", 36),
    new Persona("Pam", 25, "Programmer", 35),
    new Persona("Ted", 32, "Tester", 40));

public void conta(String ruolo) {
    System.out.print("Hanno ruolo " + ruolo + ": ");
    long c = team.stream()
        .filter(p -> p.getRuolo().equals(ruolo))
        .peek(p -> System.out.print(p.getNome() + ", "))
        .count();
    System.out.println("\nCi sono " + c + " " + ruolo);
}

// Output
// Hanno ruolo Programmer: Claire, Ed, Pam,
// Ci sono 3 Programmer

// chiamante
conta("Programmer");
```

6

Prof. Tramontana - Maggio 2020

Operazione Su Ruoli

- Data una lista di istanze di Persona, stampare i ruoli presenti e per ciascun ruolo la lista delle persone aventi quel ruolo

```
public void scriviRuoli() {
    team.stream()
        .map(p -> p.getRuolo())
        .distinct()
        .peek(r -> System.out.print("\nRuolo " + r + ": "))
        .forEach(r -> team.stream()
            .filter(p -> p.getRuolo().equals(r))
            .forEach(p -> System.out.print(p.getNome() + " ")));
}

// Output
// Ruolo Architect: Al
// Ruolo Programmer: Claire Ed Pam
// Ruolo Tester: Ted
```

7

Prof. Tramontana - Maggio 2020

Classe Pagamento

```
public class Pagamento {
    private Persona pers;
    private int importo;

    public Pagamento(Persona p, int v) {
        pers = p;
        importo = v;
    }

    public Persona getPers() {
        return pers;
    }

    public int getImporto() {
        return importo;
    }
}
```

8

Prof. Tramontana - Maggio 2020

Lista Di Pagamenti

- Data una lista di nomi di persona, creare la lista di istanze di Pagamento con il costo calcolato in base a ciascuna persona, e stampare i pagamenti

```
private List<String> daPagare = List.of("Pam", "Ed", "Ted");

private List<Pagamento> pagati = new ArrayList<>();

public void pagamenti() {
    pagati = team.stream()
        .filter(p -> daPagare.contains(p.getNome()))
        .map(p -> new Pagamento(p, p.getCosto() * 30))
        .peek(v -> System.out.print(v.getPers().getNome() +
                                   " " + v.getImporto() + " "))
        .collect(Collectors.toList());
}

// Output
// Ed 1080 Pam 1050 Ted 1200
```

9

Prof. Tramontana - Maggio 2020

Lista Di Buste Paga

- Data una lista di istanze di Persona, creare una lista con istanze di BustaPaga con l'importo calcolato in base al costo di ciascuna persona, e ordinare la lista per nome persona

```
private List<BustaPaga> buste;

public void generaBustePaga() {
    buste = team.stream()
        .map(p -> new BustaPaga(p))
        .peek(b -> b.calcolaCostoBase())
        .peek(b -> b.aggiungiBonus())
        .sorted(Comparator.comparing(b -> b.getPersona().getNome()))
        .collect(Collectors.toList());
}
```

10

Prof. Tramontana - Maggio 2020

```
public class BustaPaga {
    private Persona pers;
    private int totale;

    public BustaPaga(Persona p) {
        pers = p;
    }

    public void calcolaCostoBase() {
        totale = pers.getCosto() * 30;
    }

    public void aggiungiBonus() {
        totale = (int) Math.round(totale * 1.1);
    }

    public Persona getPersona() {
        return pers;
    }

    public void stampa() {
        System.out.println(pers.getNome() + "\t " + totale + " euro");
    }

    public int getImporto() {
        return totale;
    }
}
```

11

Prof. Tramontana - Maggio 2020

Lista Di Busta Paga

- Data la lista di istanze di BustaPaga, stampare il nome di ciascuna persona e l'importo e calcolare la somma degli importi

```
public int calcolaSomma() {
    return buste.stream()
        .peek(b -> b.stampa())
        .mapToInt(b -> b.getImporto())
        .sum();
}

// Output
// Al      1452 euro
// Claire   1254 euro
// Ed       1188 euro
// Pam      1155 euro
// Ted      1320 euro
// Totale:  6369 euro

// chiamante
System.out.println("Totale:\t " + calcolaSomma() + " euro");
```

12

Prof. Tramontana - Maggio 2020