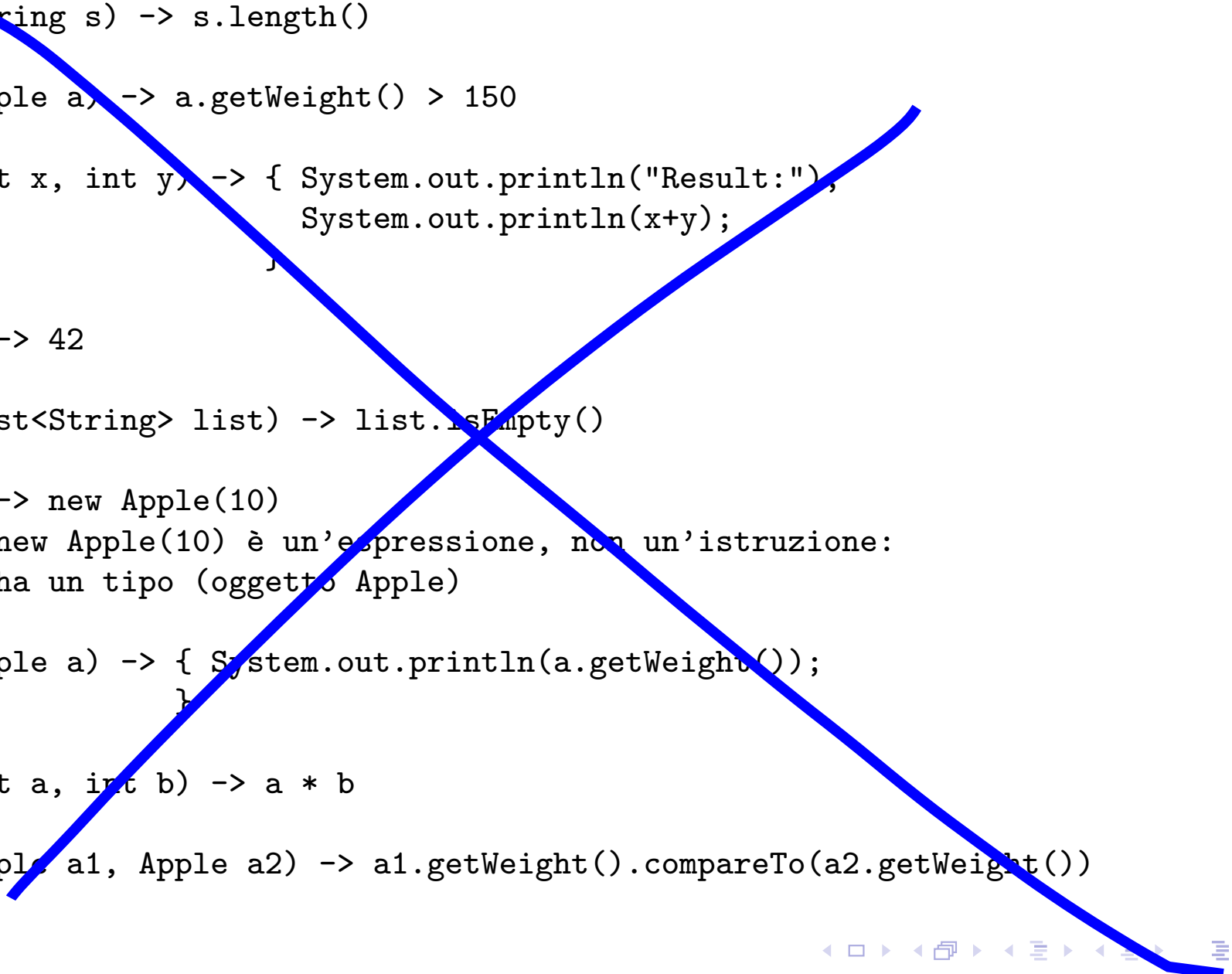
LPP - +3CFU (corso da 9 CFU)

Ancora sulle lambda espressioni di Java 8

Viviana Bono

Esempi di lambda espressioni

```
(String s) -> s.length()

(Apple a) -> a.getWeight() > 150

(int x, int y) -> { System.out.println("Result:");
                    System.out.println(x+y);
                  }

() -> 42

(List<String> list) -> list.isEmpty()

() -> new Apple(10)
// new Apple(10) è un'espressione, non un'istruzione:
// ha un tipo (oggetto Apple)

(Apple a) -> { System.out.println(a.getWeight());
             }

(int a, int b) -> a * b

(Apple a1, Apple a2) -> a1.getWeight().compareTo(a2.getWeight())
```



Qualche interfaccia funzionale (annotation: @FunctionalInterface)

```
public interface Predicate<T> {
  boolean test (T t);
}

public interface Comparator<T> {
  int compare(T o1, T o2);
}

public interface Runnable {
  void run();
}

public interface Callable<V> {
  V call();
}

public interface Function<T,R> {
  R apply(T t);
}
```

Quali lambda espressioni sono usate correttamente? (Consultare la slide precedente)

1. SI

```
public void execute(Runnable r){
    r.run();
}

execute(() -> {});
```

2. SI

```
public Callable<String> fetch() {
    return () -> "Tricky example ;-)";
}
```

3. NO

```
Predicate<Apple> p = (Apple a) -> a.getWeight();
```

(la lambda restituisce un int, non un boolean)

Functional interface API (Package java.util.function)

https://docs.oracle.com/javase/8/docs/api/java/util/function/package-summary.html

Le lambda e le variabili nel loro body

```
interface I{
  int id(int x);
}

// main
/** Le variabili usate nelle lambda espressioni devono essere final
    (dichiarate così) o effectively final (non modificate né dentro
    la lambda, né dopo). Nel codice di seguito, la linea relativa a
    inc2 non compila a causa della modifica su c:
 */

 int c = 0;
 I inc1 = (int x)->{System.out.println(c); return c;};
 // I inc2 = (int x)->{System.out.println(c); c++; return c;}; // NON COMPILA!
 inc1.id(3);
 // inc2.id(4);

 /** Razionale: è stato deciso di avere il body delle lambda il
     più possibile vicino al paradigma funzionale, quindi di evitare
     modifiche dirette alla memoria durante l'esecuzione di una
     lambda.
  */
```

Le lambda e la heap

```
interface I{
  int id(int x);

}
```

```
class Count{
   public int count;
   public Count(int c) {count = c;}
}
```

```
// main
 /** Si possono però cambiare i campi di un oggetto, poiché
     non si modifica la variabile sullo stack, ma una cella della heap:
  */
 Count co = new Count(0);
 I inc1 = (int x) -> {System.out.println(co.count); return co.count;};
 I inc2 = (int x) -> {
   System.out.println(co.count);
   co.count++;
   System.out.println(co.count);
   return co.count;
  };
 inc1.id(3);
 inc2.id(4);

// ATTENZIONE: cambiare la heap dentro i lambda
// può portare a errori di programmazione subdoli,
// è quindi da evitare!
```

Sulle closure

```
interface I{
  int id(int x);
}

// main
 I var = (int x) -> x;
 I clo1 = (int x) -> {return var.id(9);};
 System.out.println(clo1.id(1));
 // stampa 9, perche'?
```