

# Programmazione I-B 2020-21

Laboratorio T2

Attilio Fiandrotti

[attilio.fiandrotti@unito.it](mailto:attilio.fiandrotti@unito.it)

29 Ottobre 2020

# Outline

- Soluzione esercizi 22 Ottobre
  - Libreria funzioni matematiche
  - Libreria funzioni logiche
  - Libreria funzioni grafiche
- Nuovi esercizi
  - Cambiamonete
  - Crescita esponenziale
  - Serie finite

# Soluzione esercizi 22 Ottobre

# Esercizio: Libreria di funzioni aritmetiche

Si sviluppi una classe che implementi le funzioni aritmetiche somma, sottrazione, moltiplicazione, divisione, esponenziale, resto. Per scelta, i metodi siano definiti senza rifarsi ad operatori Java built-in (es: +, -, \*, /, ...) ma utilizzando le funzioni di assegnamento, incremento e decremento 1, salto condizionato effettivamente disponibili in alcuni microcontrollori. Si sviluppi quindi l'opportuna classe di test e si collaudi la classe sviluppata.

# Libreria di funzioni aritmetiche – motivazione

- Didattica: verificare che con semplici primitive come incremento/decremento 1 e salto condizionato é possibile implementare operazioni complesse come la moltiplicazione
- Pratica: programmazione microcontrollori con ISA elementare

# Libreria di funzioni aritmetiche – piu(a,b)

- Approccio: sommo a *res* il valore 1 un numero di volte *b*

- Memorizzo i risultati parziali nell'*accumulatore* *res*

$$res = a + \sum_{i=b}^{i=1} 1$$

- La variabile ausiliaria *i* é il *contatore di cicli* (*b* cicli totali)

```
public static int piu(int a, int b) {  
    int res = a;  
    int i = b;  
    while (i > 0) {  
        res = res + 1;  
        i = i - 1;  
    }  
    return res;  
}
```

# Libreria di funzioni aritmetiche – piu(a,b)

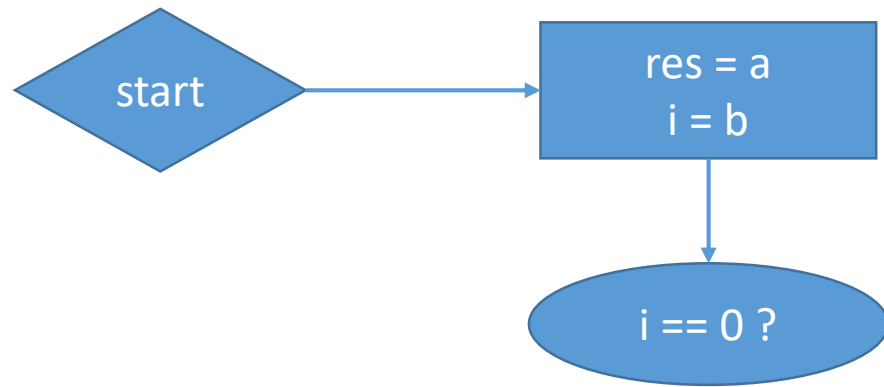


Inizializzazione variabili:  
res: *accumulatore* incrementi ad *a*  
i: *contatore* di cicli (*b* cicli totali)

```
public static int piu(int a, int b) {  
    int res = a;  
    int i = b;  
    while (i > 0) {  
        res = res + 1;  
        i = i - 1;  
    }  
    return res;  
}
```

The initialization lines 'int res = a;' and 'int i = b;' are circled in red in the original image.

# Libreria di funzioni aritmetiche – piu(a,b)



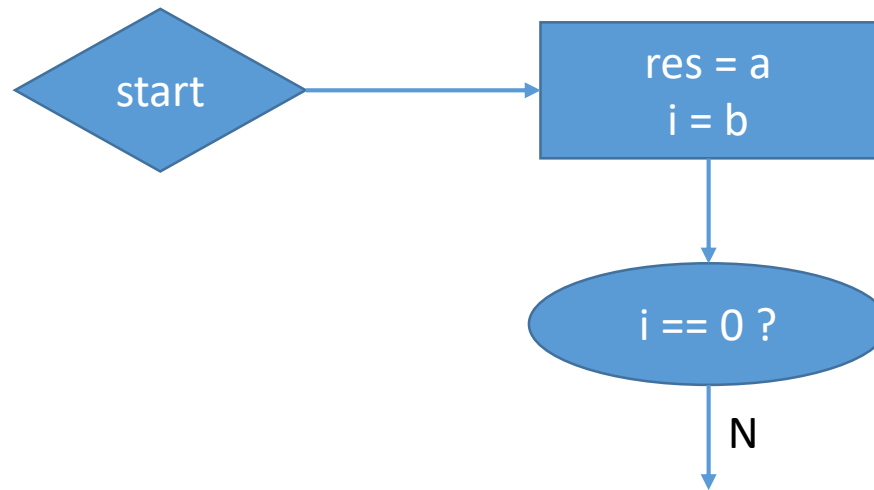
Controllo ingresso ciclo:

```
public static int piu(int a, int b) {  
    int res = a;  
    int i = b;  
    while (i > 0) {  
        res = res + 1;  
        i = i - 1;  
    }  
    return res;  
}
```

A red dashed oval highlights the `while (i > 0)` loop in the code. A blue arrow from the text "Controllo ingresso ciclo:" points to the condition `i > 0` inside the loop.



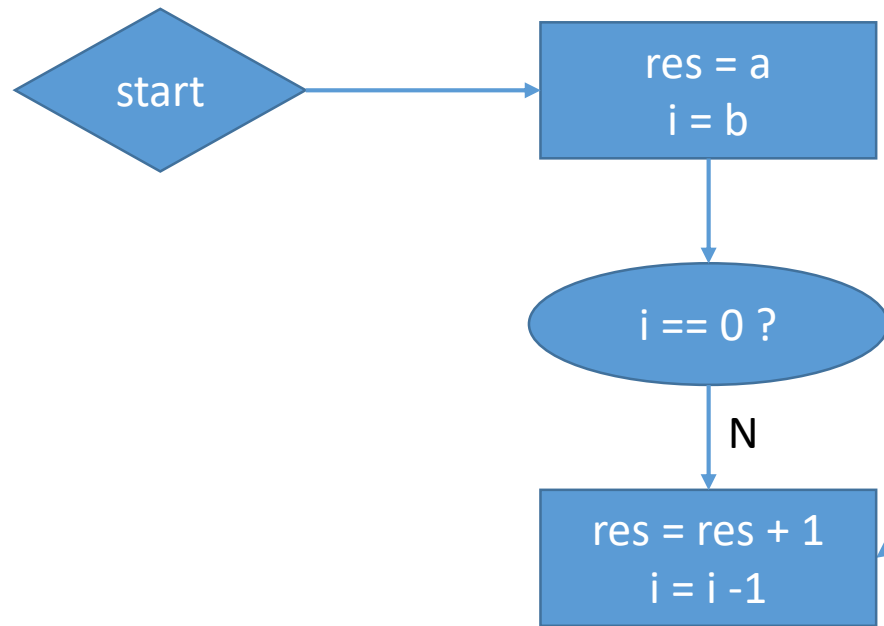
# Libreria di funzioni aritmetiche – piu(a,b)



Controllo ingresso ciclo:  
Eseguo ciclo se num cicli residui  $i > 0$

```
public static int piu(int a, int b) {  
    int res = a;  
    int i = b;  
    while (i > 0) {  
        res = res + 1;  
        i = i - 1;  
    }  
    return res;  
}
```

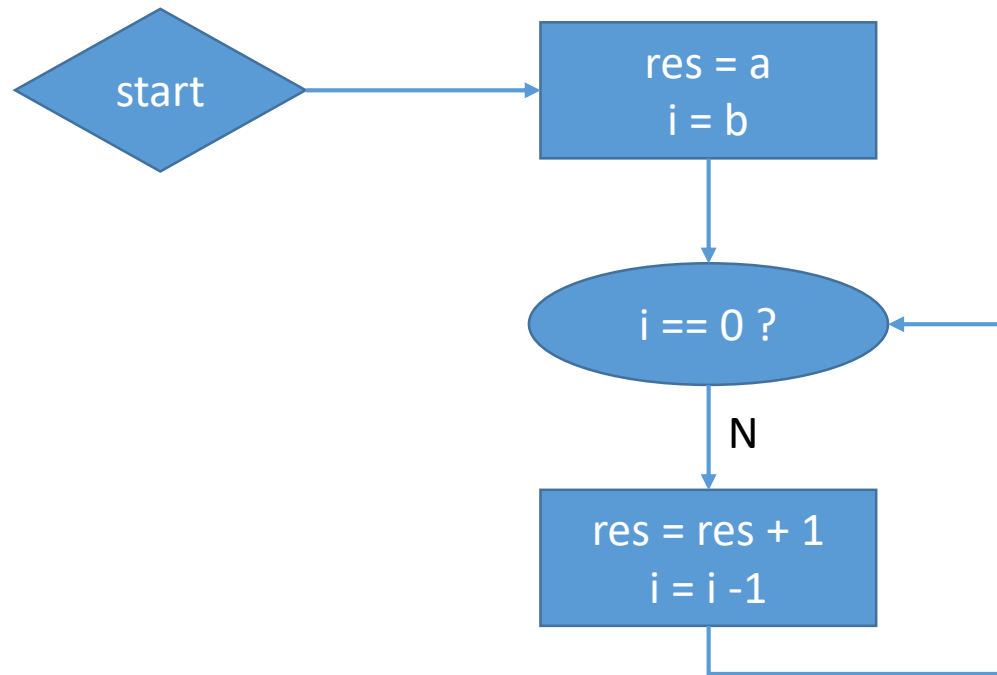
# Libreria di funzioni aritmetiche – piu(a,b)



Corpo del ciclo:  
incremento accumulatore res  
Decremento contatore cicli residui *i*

```
public static int piu(int a, int b) {  
    int res = a;  
    int i = b;  
    while (i > 0) {  
        res = res + 1;  
        i = i - 1;  
    }  
    return res;  
}
```

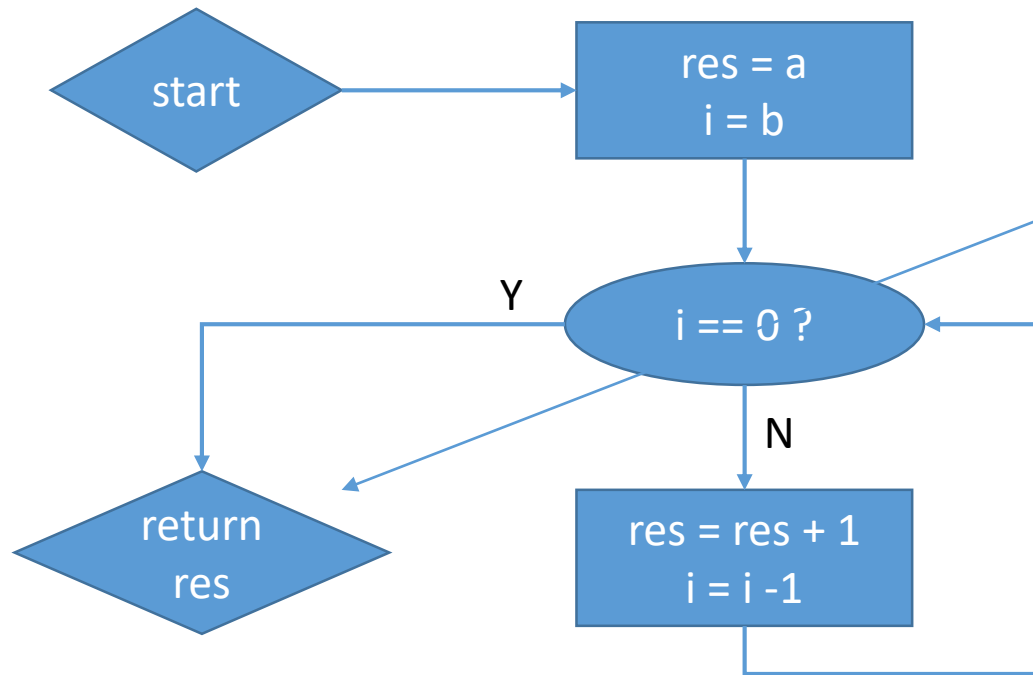
# Libreria di funzioni aritmetiche – piu(a,b)



Salto all'indietro al controllo di ciclo

```
public static int piu(int a, int b) {  
    int res = a;  
    int i = b;  
    while (i > 0) {  
        res = res + 1;  
        i = i - 1;  
    }  
    return res;  
}
```

# Libreria di funzioni aritmetiche – piu(a,b)



Esco dal ciclo e ritorno *res*  
Avrò eseguito *b* cicli

```
public static int piu(int a, int b) {  
    int res = a;  
    int i = b;  
    while (i > 0) {  
        res = res + 1;  
        i = i - 1;  
    }  
    return res;  
}
```

# Libreria di funzioni aritmetiche – piu(a,b)

- Approccio: sommo ad *acc* il valore 1 un numero di volte *b*
  - Memorizzo i risultati parziali in *a* riutilizzata come *accumulatore*

$$a = a + \sum_b^1 1$$

- La variabile *b* é riutilizzata come *contatore di cicli* (*b* cicli totali)

```
public static int piuIncNonConservativa(int a, int b) {  
    while (b > 0) {  
        a = a + 1;  
        b = b - 1;  
    }  
    return a;  
}
```

Riuso delle variabili locali *a*, *b*

# Libreria di funzioni aritmetiche – per(a,b)

- Approccio: sommo a *res* il valore *a* un numero *b* volte
  - Memorizzo i risultati parziali nell'*accumulatore* *res*

$$res = \sum_{i=1}^b a$$

- La variabile ausiliaria *i* é il *contatore di cicli* (*b* cicli totali)

```
public static int perDecrescente(int a, int b) {  
    int res = 0;  
    int i = b;  
    while (i > 0) {  
        res = piu(res, a);  
        i = i - 1;  
    }  
    return res;  
}
```

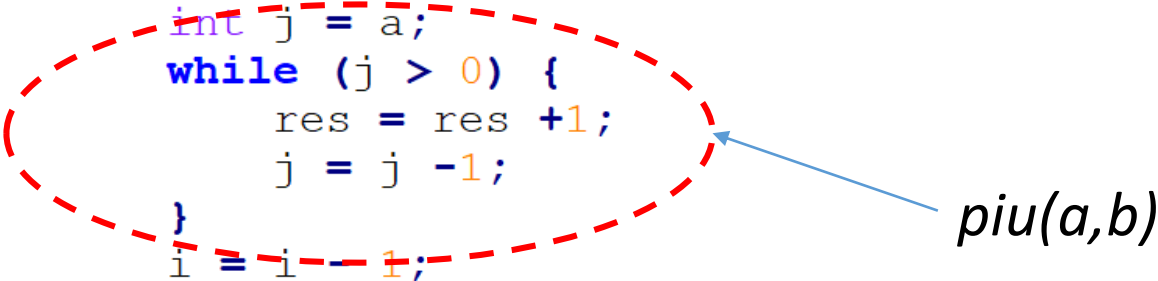
riutilizzo metodo *piu(a,b)*

# Libreria di funzioni aritmetiche – per(a,b)

- Approccio: sommo a *res* il valore 1 un numero  $a*b$  volte
  - Loop esterno su *i* per moltiplicazione
  - Loop interno su *j* per somma

$$res = \sum_{i=b}^{i=1} \sum_{j=a}^{j=1} 1$$

```
public static int perDecrescente(int a, int b) {  
    int res = 0;  
    int i = b;  
    while (i > 0) {  
        int j = a;  
        while (j > 0) {  
            res = res + 1;  
            j = j - 1;  
        }  
        i = i - 1;  
    }  
    return res;  
}
```



# Libreria di funzioni aritmetiche – diviso(a,b)

- Approccio: *conto* quante volte posso decrementare  $a$  di  $b$  prima che il *resto* diventi negativo

```
public static int div(int a, int b) {  
    //Nota: resto superfluo rispetto ad a  
    int resto = a;  
    int cnt = 0;  
    while (resto >= b) {  
        resto = meno(resto, b);  
        cnt = cnt + 1;  
    }  
    return cnt;  
}
```



# Libreria di funzioni aritmetiche – resto(a,b)

- Approccio: *conto* quante volte posso decrementare *a* di *b* prima che il *resto* diventi negativo e restituisco il resto

```

                                resto
public static int div(int a, int b) {
    //Nota: resto superfluo rispetto ad a
    int resto = a;
    int cnt = 0;
    while (resto >= b) {
        resto = meno(resto, b);
        cnt = cnt + 1;
    }
    return cnt resto;
}

```

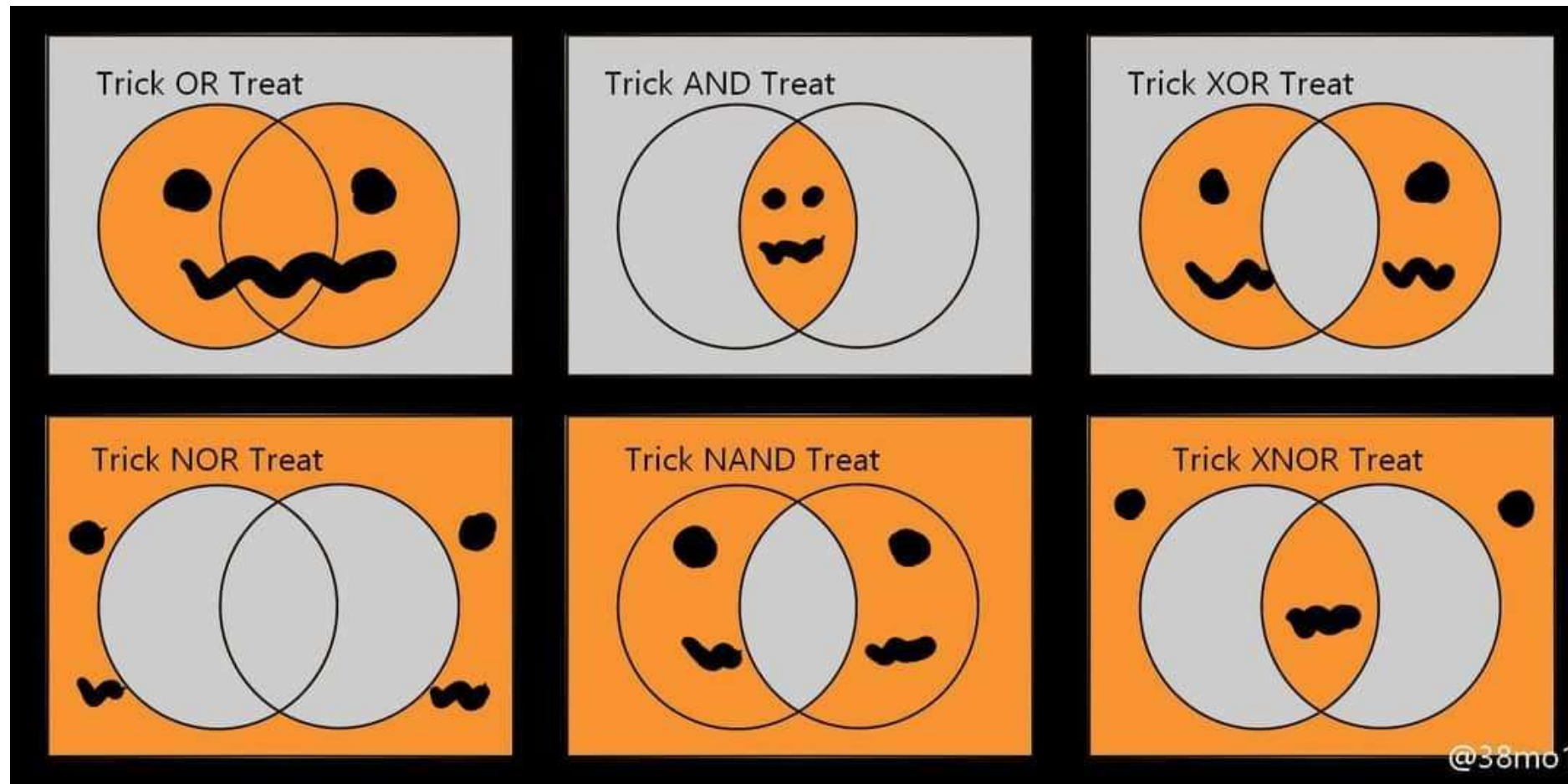
# Esercizio: libreria di funzioni booleane

Si sviluppi una classe che implementi i metodi che realizzano le 16 funzioni booleane a due argomenti specificate nel file *201022-050-Tabelle-di-verita.java*.

Per scelta, i metodi siano definiti usando in maniera essenziale solo i test annidati necessari a produrre il risultato senza rifarsi ad operatori Java built-in (es: `&&`, `//`, `!`).

Si sviluppi quindi l'opportuna classe di test e si collaudi la classe sviluppata.

# Esercizio: libreria di funzioni booleane

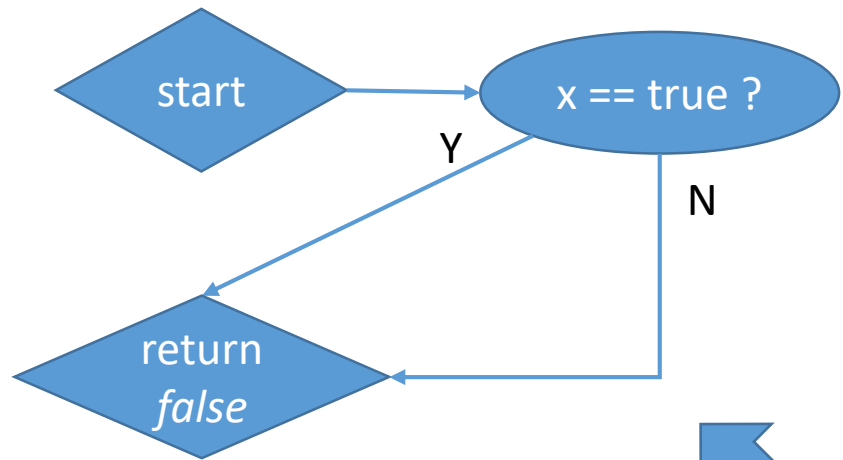


# Libreria di funzioni booleane – motivazione

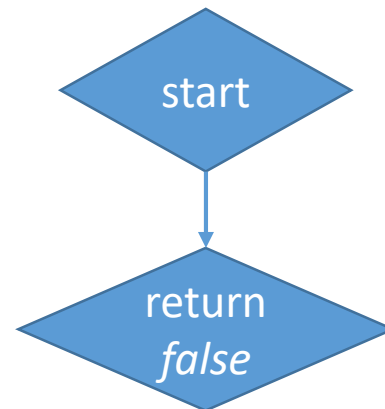
- Didattica: verificare che con salti condizionati annidati su una *singola* variabile booleana é possibile implementare qualsivoglia funzione logica
- Pratica: programmazione architetture ISA elementari come IJVM (vedi corso *Architetture di elaborazione*, programmazione livello ISA e MAL)
- Approccio: annidare i salti condizionati per implementare la tavola di verità di ogni funzione

# Libreria di funzioni booleane – zero(x,y)

	X		zero	
	-	+	---	
	F		T	
	T		F	

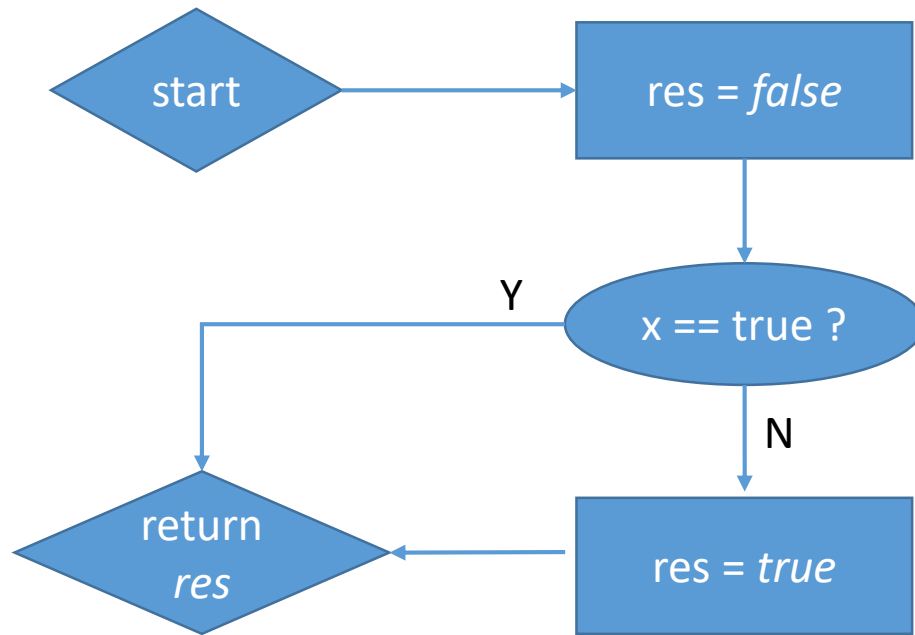


```
public static boolean zero(boolean x, boolean y) {  
    return false;  
}
```



# Libreria di funzioni booleane – not(x)

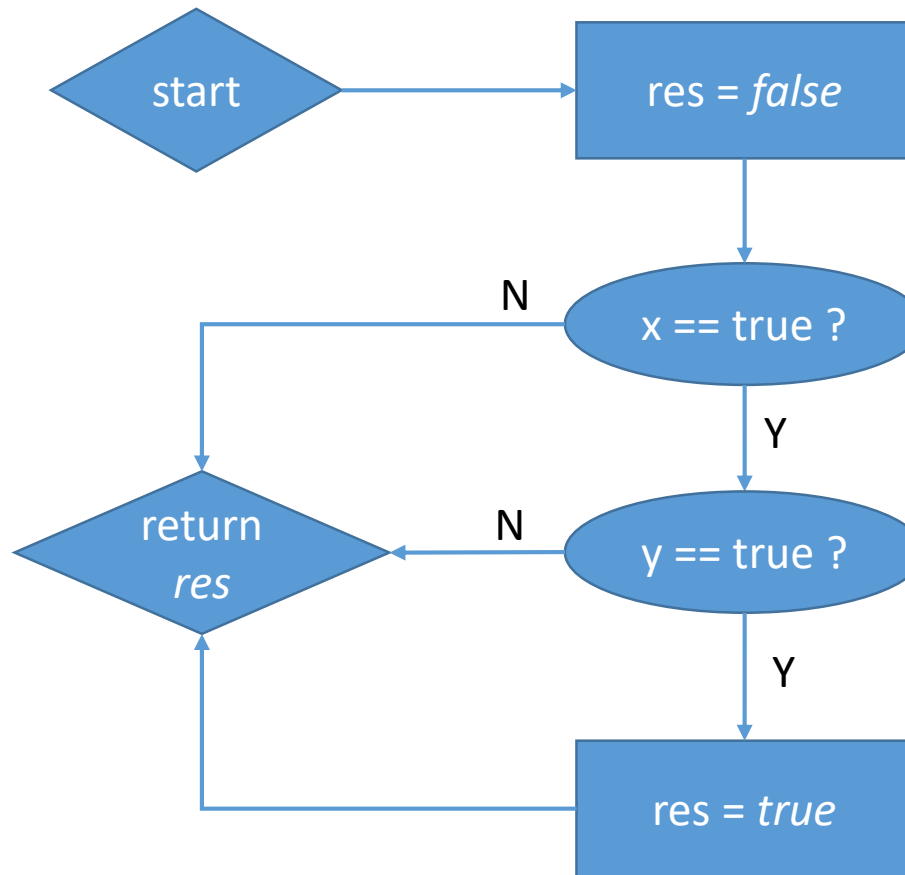
	X		zero	
	-	+	---	
	F		F	
	T		F	



```
public static boolean not(boolean x) {  
    boolean res = false;  
    if(x) {  
    } else {  
        res = true;  
    }  
    return res;  
}
```

# Libreria di funzioni booleane – and(x,y)

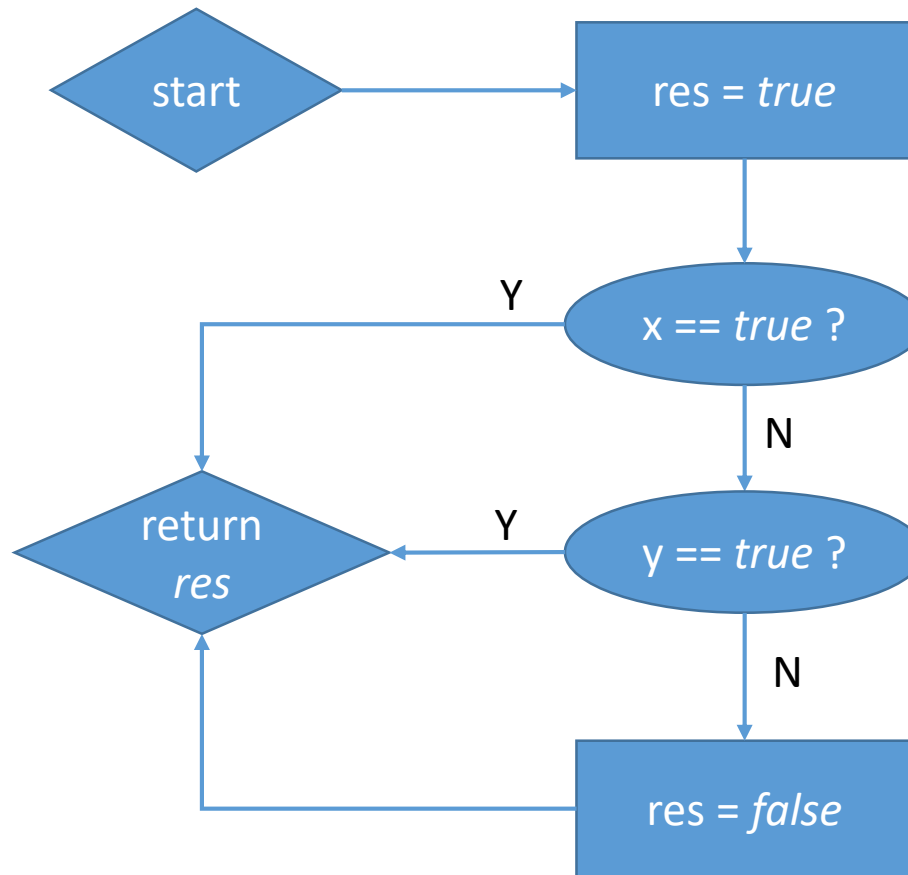
X	Y	X and Y
-	-	-----
F	F	F
F	T	F
T	F	F
T	T	T



```
public static boolean and(boolean x, boolean y)
{
    boolean res = false;
    if(x) {
        if(y) {
            res = true;
        }
    }
    return res;
}
```

# Libreria di funzioni booleane – or(x,y)

X	Y	X or Y
-	-	-----
F	F	F
F	T	T
T	F	T
T	T	T

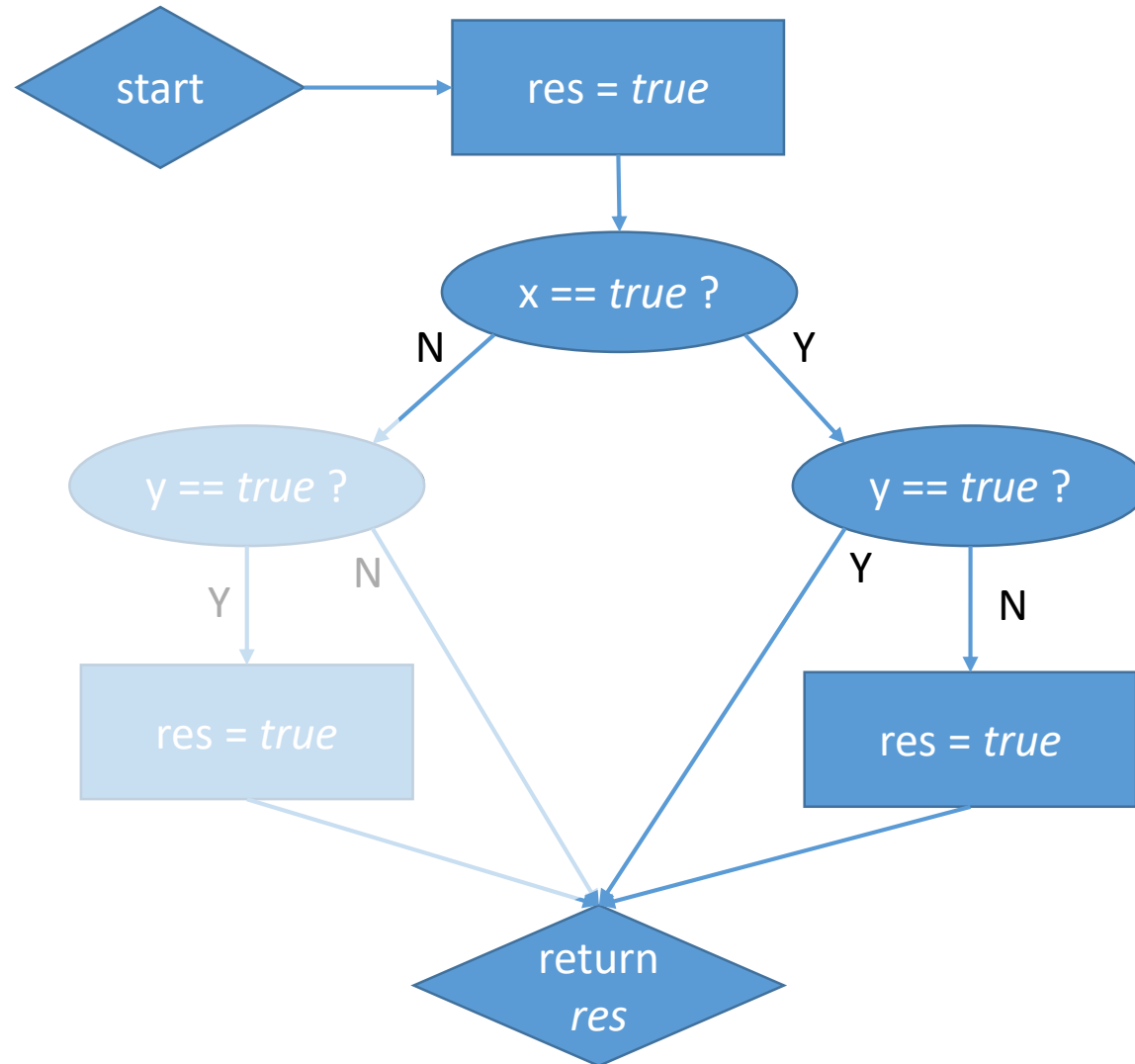


```
public static boolean or(boolean x, boolean y) {  
    boolean res = true;  
    if(x) {  
    } else {  
        if(y) {  
        } else {  
            res = false;  
        }  
    }  
    return res;  
}
```



# Libreria di funzioni booleane – xor(x,y)

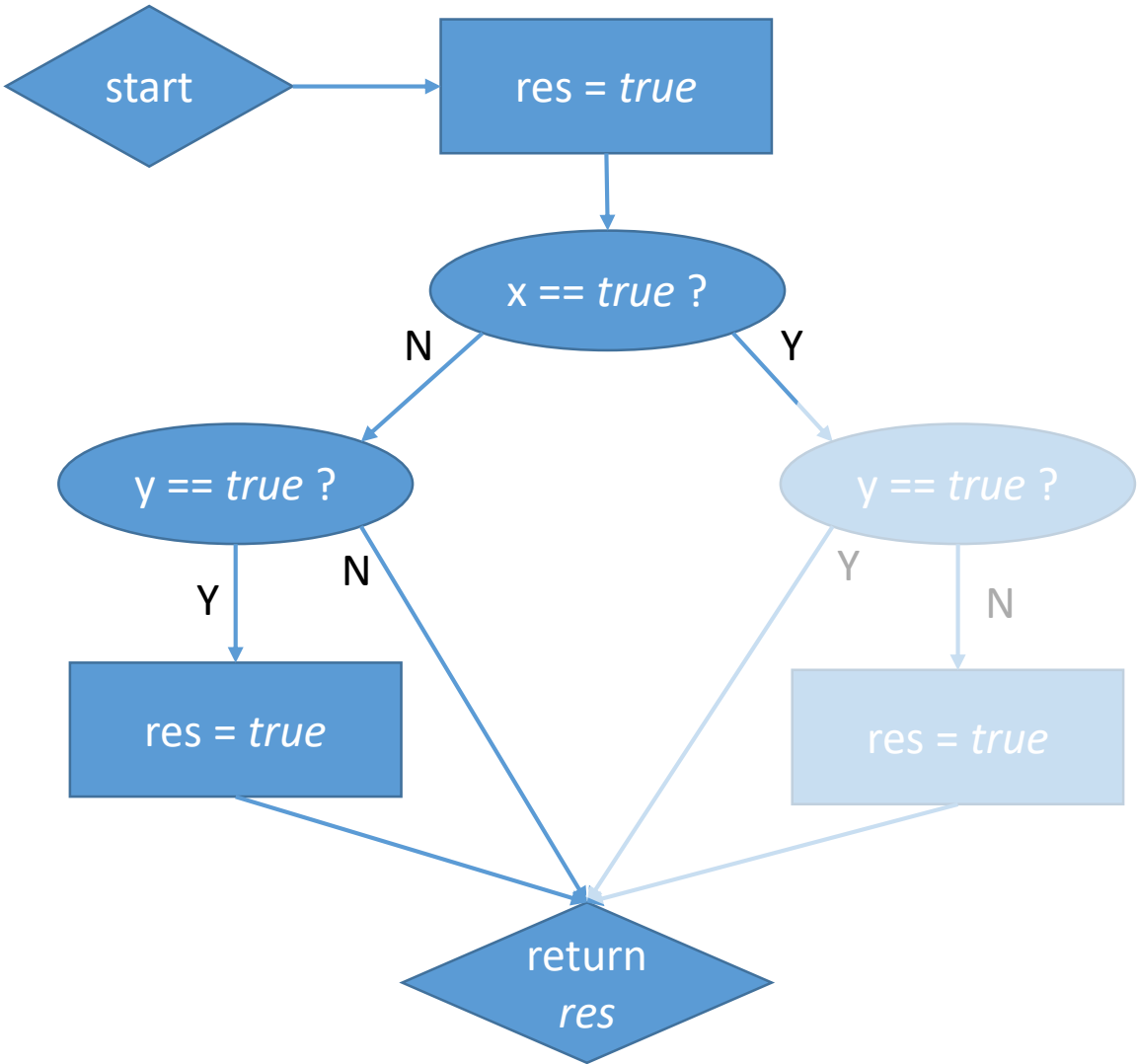
X	Y	X xor Y
-	-	-----
F	F	F
F	T	T
T	F	T
T	T	F



```
public static boolean xor(boolean x, boolean y) {
    boolean res = false;
    if(x) {
        if(y) {
        } else {
            res = true;
        }
    } else {
        if(y) {
            res = true;
        }
    }
    return res;
}
```

X	Y	X xor Y
-	-	-----
F	F	F
F	T	T
T	F	T
T	T	F

# Libreria di funzioni booleane – xor(x,y)



```
public static boolean xor(boolean x, boolean y) {
    boolean res = false;
    if(x) {
        if(y) {
        } else {
            res = true;
        }
    } else {
        if(y) {
            res = true;
        }
    }
    return res;
}
```

# Libreria di funzioni booleane – Testing

```
public class BooleanFinaleTest {  
  
    public static void main(String[] args) {  
        System.out.println("---- notTest");  
        notTest();  
        System.out.println("---- andTest");  
        andTest();  
    }  
  
    public static void notTest() { // operatore Java built-in: !x  
        System.out.println(false==BooleanFinale.not(true));  
        System.out.println(true==BooleanFinale.not(false));  
    }  
  
    public static void andTest() { // operatore Java built-in: x&&y  
        System.out.println(true==BooleanFinale.and(true,true));  
        System.out.println(false==BooleanFinale.and(false,true));  
        System.out.println((false&&true)==BooleanFinale.and(false,true));  
    }  
}
```

# Esercizio: libreria di funzioni grafiche

Scrivere una classe `Asterischi.java` e la corrispondente `AsterischiTest.java`. `Asterischi.java` contiene metodi con le seguenti caratteristiche

- `aCapo()` con l'ovvio significato d'andare a capo
- `riga(int n, char c)` -> stampa una riga con  $n$  copie del carattere  $c$

ESEMPIO: `riga(10, '*')` stamperà:

```
>java Asterischi
*****
```

- `rettangolo(int n, int m, char c)` -> stampa un rettangolo con  $n*m$  copie del carattere  $c$ .

ESEMPIO: `rettangolo(5,7, '*')` stamperà:

```
>java Asterischi
*****
*****
*****
*****
*****
```

# Esercizio: libreria di funzioni grafiche

- Approccio:
- Definire innanzitutto le primitive grafiche più elementari
  - Carattere -> Linea-> Figura (serie di linee)
- Combinare le primitive in funzioni più complesse

# Libreria di funzioni grafiche - esempio

```
/* Stampa carattere speciale "newline" (CR+)LF */
public static void aCapo() {
    System.out.println();
}
/* Riga con n copie del carattere c*/
public static void riga(int n, char c) {
    int i = 0;
    while(i < n) {
        System.out.print(c);
        i = i + 1;
    }
}
/* Rettangolo con n x m copie del carattere c*/
public static void rettangolo(int n, int m, char c) {
    int i = 0;
    while(i < n - 1) {
        riga(m,c); aCapo();
        i = i + 1;
    }
    if (n > 0)
        riga(m,c);
}
```

# Esercizi settimana 26-30 Ottobre

# Esercizio: macchina cambiamonete

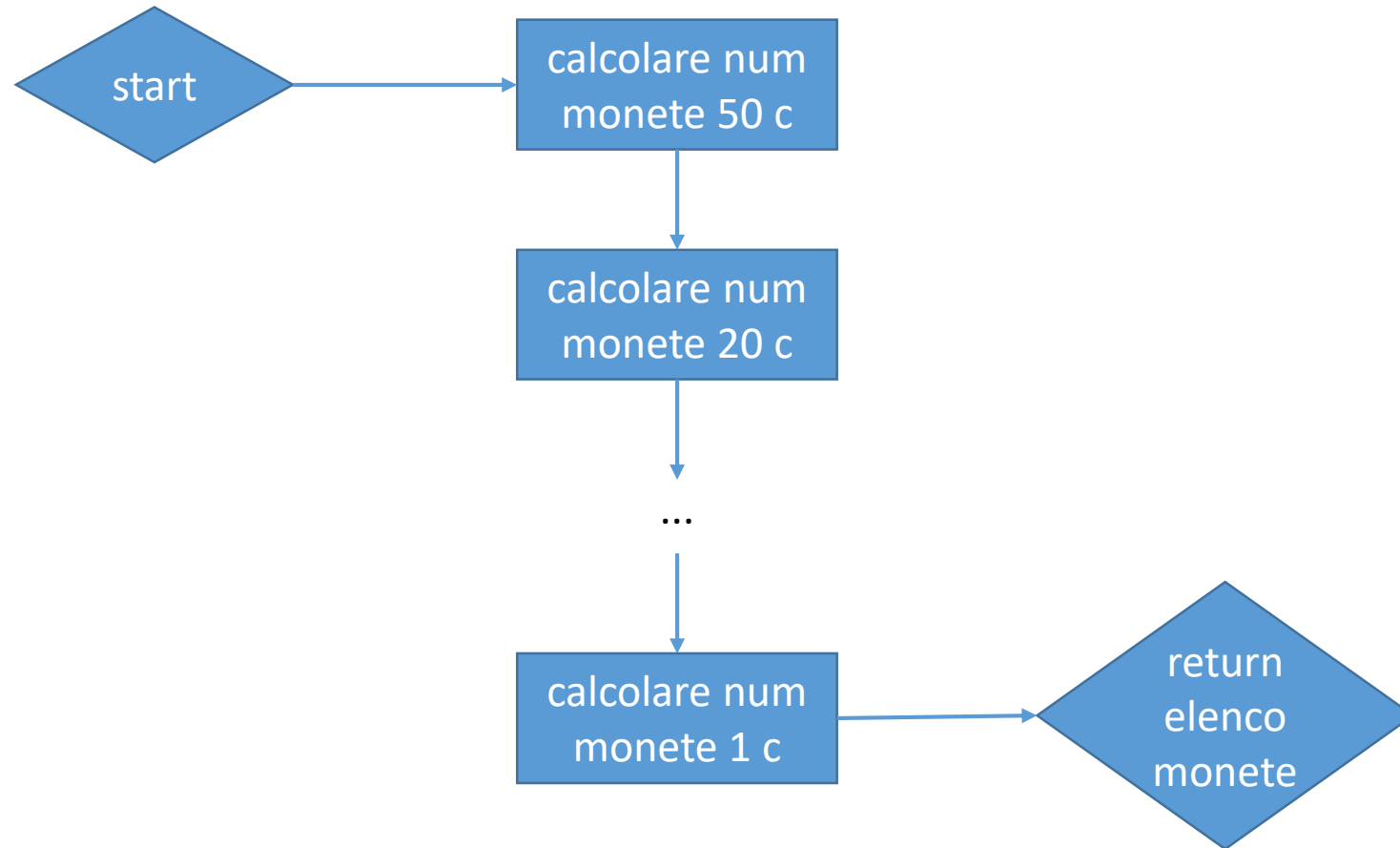
- Implementare un algoritmo *cambia()* che implementi una macchina cambia monete. La macchina accetta somme fino a 1 Euro e restituisce la stessa somma in monete da 50, 20, 10, 5, 2 e 1 cents. La macchina cerca di soddisfare la richiesta dell'utente resituendo il minor numero possibile di monete. Si assuma che la macchina disponga internamente di una quantità illimitata di monete da restituire all'utente.  
Esempio: se l'utente inserisce una somma pari a 45 cents, la macchina restituirà 2 monete da 20 cents e una moneta da 5 cents.
- Si sviluppi l'opportuna classe di test in modo che l'output della macchina possa essere verificato *atomicamente*.



# Macchina cambiamonete - approccio

- Risolvere il problema sequenzialmente per ogni taglia di moneta
  - Per ogni taglia ritornare il massimo numero di monete erogabili per non eccedere la quantità di denaro da erogare residua
  - La quantità di denaro residua (*if any*) sarà erogata in monete del taglio più piccolo e così via fino all'erogazione completa della somma introdotta

# Macchina cambiamonete - approccio



# Macchina cambiamonete - approccio

- Tipo di dato ritornato e test case
  - Quale tipo di dato dovrà ritornare il metodo perché il metodo possa essere verificato atomicamente ?
  - L'algoritmo dovrà restituire al metodo chiamante l'*elenco* del numero di monete erogate all'utente per ogni taglia di moneta
  - Suggerimento: utilizzare la classe *String*

# La classe String

- La classe String é un tipo Java *primitivo* per la manipolazione del testo
- String mette a disposizione operatori e metodi per le operazioni più comuni fra sequenze di caratteri

```
/* Dichiarazione di stringa */  
String sUno = "hello";  
String sDue = "world";  
  
/* Concatenazione */  
String sTre = sUno + " " + sDue + "!";  
  
/* Confronto */  
boolean res = sUno.equals(sDue);
```

# Esercizio: macchina cambiamonete

- Implementare un algoritmo *cambia()* che implementi una macchina cambia monete. La macchina accetta somme fino a 1 Euro e restituisce la stessa somma in monete da 50, 20, 10, 5, 2 e 1 cents. La macchina cerca di soddisfare la richiesta dell'utente resituendo il minor numero possibile di monete. Si assuma che la macchina disponga internamente di una quantità illimitata di monete da restituire all'utente.  
Esempio: se l'utente inserisce una somma pari a 45 cents, la macchina restituirà 2 monete da 20 cents e una moneta da 5 cents.
- Si sviluppi l'opportuna classe di test in modo che l'output della macchina possa essere verificato *atomicamente*.

# Esercizio: infestazione di scarafaggi

- Una città é infestata dagli scarafaggi. La popolazione di scarafaggi aumenta con un tasso di crescita settimanale del 95%. Nota la popolazione iniziale di scarafaggi che infestano una casa e suo il volume, si sviluppi un algoritmo che calcoli il numero di settimane dopo cui la casa sarà piena di scarafaggi. Si assuma che ogni scarafaggio abbia un volume di  $0.8 \text{ cm}^3$ .
- Si sviluppi l'opportuna classe di test per tale algoritmo.

# Infestazione di scarafaggi - approccio

- Calcolare il numero di scarafaggi presenti nella casa alla fine di ogni settimana
  - Come calcolare il numero di scarafaggi nella casa alla *fine* della settimana dato il numero iniziale ?
- Calcolare il volume occupato dagli scarafaggi alla *fine* della settimana
  - Come calcolare tale volume ?
- Il numero di scarafaggi nella casa all'inizio della settimana  $n$  é pari al numero di scarafaggi alla fine della settimana precedente  $n-1$

# Esercizio: serie numeriche (1)

- Di seguito sono elencate serie numeriche ed il valore cui convergono

$$\sum_{k=0}^n q^k = \frac{1 - q^{n+1}}{1 - q} \quad (1)$$

$$\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6} \quad (2)$$

$$\sum_{k=0}^{\infty} \frac{1}{(2k+1)^2} = \frac{\pi^2}{8} \quad (3)$$

$$\sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1} = \frac{\pi}{4} \quad (4)$$

$$\sum_{k=0}^{\infty} \frac{z^k}{k} = e^z \quad (5)$$



# Esercizio: serie numeriche (2)

- Completare la classe Serie.java per le serie 2-4 (la serie 1 serve d'esempio)
- Completare la classe SerieTest.java con test sui metodi aggiunti in Serie.java secondo l'esempio fornito
- I risultati dei metodi che implementeranno le serie numeriche non produrranno necessariamente un valore esatto (la sommatoria avrà un numero finito di termini). Si sviluppi un opportuno test che verifichi un risultato quando esso è in un intervallo ritenuto ragionevole