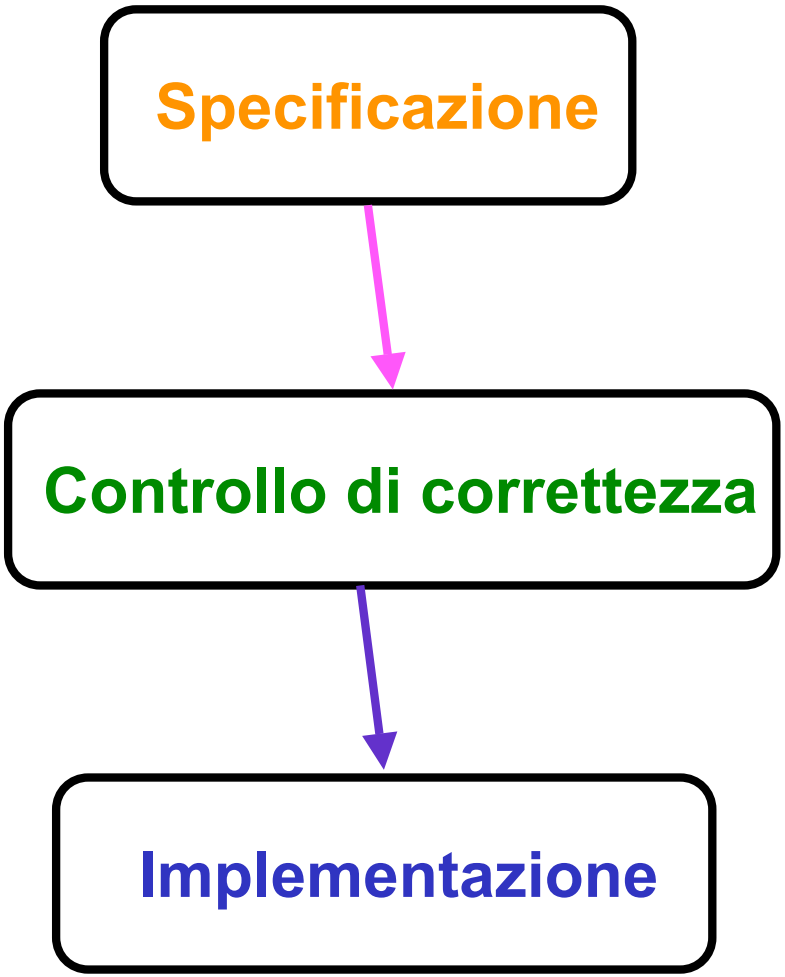
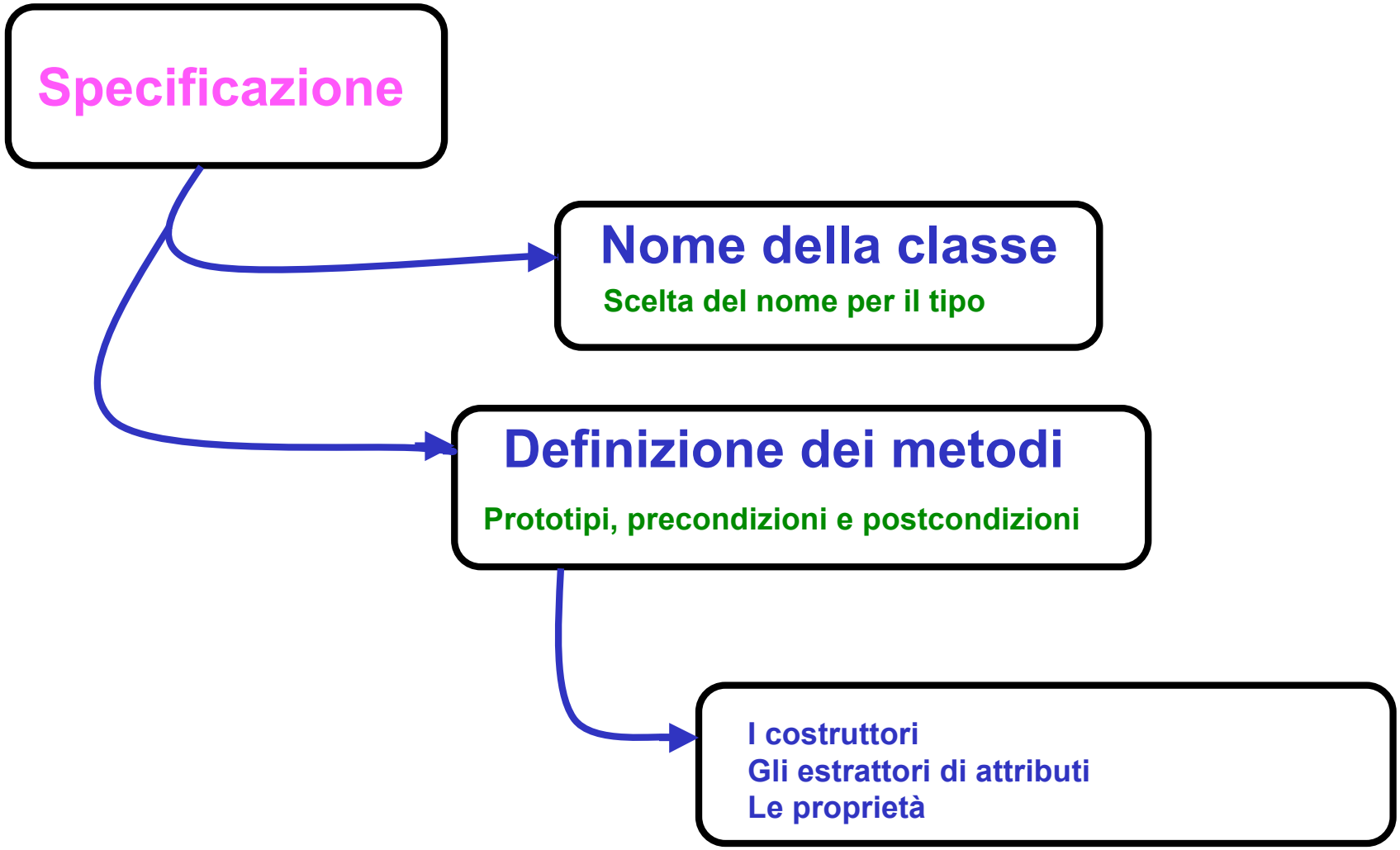
I passi fondamentali per la costruzione di una classe:




Specificazione
Nome della classe
Scelta del nome per il tipo
Definizione dei metodi
Prototipi, precondizioni e postcondizioni
I costruttori
Gli estrattori di attributi
Le proprietà

Un esempio: una classe per la gestione di taniche

Descrizione informale dei requisiti:

1. **le taniche sono destinate a contenere dei liquidi**
2. **le taniche sono di forma rettangolare, quindi sono caratterizzate da altezza, larghezza e profondità.**
3. **per costruire una tanica si devono fornire le dimensioni.**
4. **le taniche sono di plastica, le dimensioni di una tanica non possono essere modificate dopo che una tanica è stata costruita**
5. **si deve assegnare ad ogni tanica la sua capacità. Questa deve essere inferiore al suo volume, per ovvie ragioni di sicurezza.**
6. **per aggiungere liquido ad una tanica bisogna che la sua capacità non sia stata raggiunta e non bisogna in ogni caso superarla.**
7. **per ogni tanica bisogna poter sapere quanto liquido contiene e quanto liquido può ancora contenere**
8. **da una tanica si può prelevare una quantità di liquido inferiore o uguale al suo contenuto.**

```
/**** Nome del file: Tanica.h ****/

/**** Nome della classe****/
typedef struct tanica * Tanica;


/**** Metodi  della classe ****/
```

Per ogni metodo si deve

1. **decidere cosa restituisce**
2. **scegliere opportunamente il nome**
3. **stabilire quali parametri di input servono**
4. **precisare le precondizioni e**
5. **le postcondizioni**

```
/*prima i costruttori*/
TanicaP CostTan ( double alt, double larg, double prof );
/* restituisce una tanica avente le dimensioni specificate
prec: alt >0 && larg >0 && prof>0
postc: restituisce un puntatore a una tanica, NULL se non c'è spazio di
memoria per una nuova tanica*/
```

Costruttore alternativo, da utilizzare con le stesse funzioni, tranne CapMax:

```
TanicaP CostTanica (double alt, double larg, double prof,double capMax)
/* restituisce una tanica avente le dimensioni e la capacità massima
specificate
prec: alt >0 && larg >0 && prof>0 && capMax < alt*larg*prof
postc: restituisce un puntatore a una tanica,
NULL se non c'è spazio di memoria per una nuova tanica*/


/* poi i distruttori */

void DistrTan (TanicaP t );
/*libera lo spazio di memoria impegnato dal puntatore t*/
```

```
/* I metodi che restituiscono le dimensioni di una tanica */
double Alt(TanicaP t );
/*prec: t!= NULL
postc: restituisce l'altezza*/

double Larg(TanicaP t );
/*prec: t!= NULL
postc: restituisce la larghezza*/

double Prof(TanicaP t );
/*prec: t!= NULL
postc: restituisce la profondità*/
```

```
double LiqCorr(TanicaP t );
/*prec: t!= NULL
postc: restituisce la quantità di liquido contenuta*/

double DispLiq( Tank t );
/*prec: t!= NULL
postc: Restituisce  la quantità di liquido che si può aggiungere nella
tanica */

double Vol(TanicaP t);
/*prec: t!= NULL
postc: Restituisce  il volume della tanica */

int CapMax( Tank t, double cap );
/* Stabilisce la massima capacità
prec: t != NULL && cap >0
postc:  Se cap < volume della tanica restituisce vero, falso altrimenti
*/
```

```
int AggLiq( Tank t, double tot );
/* aggiunge liquido a una tanica
prec: : t != NULL && tot >0
postc: restituisce vero se  tot può essere aggiunto alla tanica,
falso altrimenti.
*/

int PrelLiq( Tank t, double tot );
/* preleva liquido da una tanica
prec: t != NULL && tot>0
postc: restituisce vero se  tot di liquido può essere prelevato,
falso altrimenti*/
```

Controllo di correttezza

Il requisito 1 è soddisfatto dalla scelta del tipo double per esprimere il contenuto.
Il requisito 2 è soddisfatto dalle scelte nel costruttore CostTan:

TanicaP CostTan (double alt, double larg, double prof);

Il requisito 3 è soddistatto perché non abbiamo alcuna funzione che possa alterare le dimensioni di una tanica, ma solo ottenerle.

1. le taniche devono contenere dei liquidi
2. le taniche sono di forma rettangolare, quindi sono caratterizzate da altezza, larghezza e profondità. Le dimensioni devono essere fornite per costruire una tanica.
3. Le taniche sono di plastica, le dimensioni di una tanica non possono essere modificate dopo che una tanica è stata costruita

Controllo di correttezza

Il requisito 4 è soddisfatto dalla funzione CapMax, la funzione Vol può essere utile ed è stata aggiunta:

```
double Vol(TanicaP t);
/*prec: t!= NULL
postc: Restituisce  il volume della tanica */

int CapMax( TanicaP t, double cap );
/* Stabilisce la massima capacità
prec: t != NULL && cap >0
postc:  Se cap < volume della tanica restituisce vero, falso
        altrimenti
*/
```

4. Si deve assegnare ad ogni tanica la sua capacità. Questa deve essere inferiore al suo volume, per ragioni di sicurezza.

Controllo di correttezza

Il requisito 5 è soddisfatto con la funzione AggLiq

5. Per aggiungere liquido ad una tanica bisogna che la sua capacità non sia stata raggiunta e non bisogna in ogni caso superarla.

```
int AggLiq( TanicaP, double tot );
/* aggiunge liquido a una tanica, se la disponibilità lo consente
prec: : t != NULL && tot >0
postc: restituisce vero se  tot è stato aggiunto alla tanica, falso
       altrimenti.
*/
```

Controllo di correttezza

Il requisito 6 è soddisfatto dalle funzioni

```
double LiqCorr(TanicaP t );
/*prec: t!= NULL
postc: restituisce la quantità di liquido contenuta*/

double DispLiq( TanicaP t );
/*prec: t!= NULL
postc: Restituisce  la quantità di liquido che si può aggiungere nella
       tanica */
```

6. Per ogni tanica bisogna poter sapere quanto liquido contiene e quanto liquido può ancora contenere

Controllo di correttezza

Il requisito 7 è soddisfatto dalla funzione PrelLiq

```
int PrelLiq( TanicaP t, double tot );
/* preleva tot di liquido da una tanica, se tot è minore del liquido contenuto
prec: t != NULL && tot>0
postc: restituisce vero se  tot di liquido è stato prelevato, falso altrimenti*/
```

7. da una tanica si può prelevare una quantità di liquido inferiore o uguale al suo contenuto

Implementazione

```
/*Nome del file: Tanica.c */

/*Prima tutte le inclusioni, da libreria*/
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

/*Poi le inclusioni particolari*/

#include "Tanica.h"

/*qui tutte le costanti simboliche*/
#define TRUE 1
#define FALSE 0
```

```
/*Definizione degli attributi*/

struct tanica {
            double alt, larg, prof;
            double cap_max;
            double quant_corr;
            };


/* Costruttore */
TanicaP CostTan (double alt, double larg, double prof)
{TanicaP t;
assert(alt>0);
assert(larg >0);
assert(prof >0);
t = malloc( sizeof(struct tanica) );
if ( t == NULL ) printf("CostTan: memoria insufficiente \n");
else
      {t->alt = alt;
        t->larg = larg;
        t->prof = prof;
        t->cap_max = t->quant_corr = 0.0;
      }
return t;
}
```

```
/* Distruttore */
void DistrTan( TanicaP t )
            {free( t );}

/* Gli estrattori di attributi */

double Alt(TanicaP t )
/*prec: t!= NULL
postc: restituisce l'altezza*/
{assert(t);
  return t->alt;
    }

double Larg(TanicaP t )
/*prec: t!= NULL
postc: restituisce la larghezza*/
{assert(t);
 return t->larg;
    }
```

```
double Prof(TanicaP t );
/*prec: t!= NULL
postc: restituisce la profondità*/
{assert(t);
 return t->prof;
 }

double LiqCorr(TanicaP t );
/*prec: t!= NULL
postc: restituisce la quantità di liquido contenuta*/
{assert(t);
return t->quant_corr;
 }

double DispLiq( Tank t );
/*prec: t!= NULL
postc: Restituisce  la quantità di liquido che si può aggiungere nella tanica */
{assert(t);
 return t->cap_max -  t->quant_corr;
 }
```

```
double Vol(TanicaP t)
/*prec: t!= NULL
postc: Restituisce  il volume della tanica */
{assert(t);
return t->alt*t->larg*t->prof;
}

int CapMax( TanicaP t, double cap );
/* Stabilisce la massima capacità
prec: t != NULL && cap >0
postc:  Se cap < volume della tanica restituisce vero, falso altrimenti
*/
{assert(t);
 assert(cap >0);
 if ( cap < (t -> alt * t -> larg * t->prof) )  /* o cap <Vol(t); ?*/
    {t->cap_max = cap;
      return TRUE;
     }
else
      return FALSE;
}
```

```
int AggLiq( TanicaP t, double tot )
/* aggiunge liquido a una tanica, se la disponibilità lo consente
prec: : t != NULL && tot >0
postc: restituisce vero se  tot è stato aggiunto alla tanica, falso altrimenti.
*/
{assert(t);
 assert(tot >0);
 if ( (t->quant_corr + tot) <= (t->cap_max) )
{t->quant_corr = t->quant_corr + tot;
                return TRUE;
                }
                else
                return FALSE;
}

int PrelLiq( TanicaP t, double tot )
/*  preleva tot di liquido da una tanica, se tot è minore del liquido contenuto
prec: t != NULL && tot>0
postc: restituisce vero se  tot di liquido è stato prelevato, falso altrimenti*/
{assert(t);
  assert(tot >0);
  if ( t->quant_corr >= tot )
                 {t->quant_corr = t->quant_corr - tot;
                return TRUE;
                }
                else
                return FALSE;
  }
```