MP 20-21 Tema 1
Sessió 5: Programació Orientada a Objectes

Classe = Dades + Operacions

RECORDEM:

Un polígon es pot definir a partir de les coordinades de cadascun dels vèrtexs que el composen.

a) Definir una estructura `Punt` per guardar les coordinades d'un vèrtex del polígon.

b) Implementar les següents funcions:

- Una funció que permeti llegir les coordinades d'un vèrtex i guardar-les en una variable que es passa com a paràmetre per referència del tipus `Punt` declarada a l'apartat anterior.
- Una funció que mostri per pantalla les coordinades d'un punt.
- Implementar una funció que calculi la distància entre dos punts:

Registre `Punt` (dades)

```
typedef struct
{
   float x;
   float y;
} Punt;
```

+

Funcions que operen amb `Punt`

```
void llegeixPunt(Punt& p);
void mostraPunt(Punt & p);
float distanciaPunts(Punt &p1, Punt &p2);
```

Classe = Dades + Operacions

Registre Punt (dades)

```
typedef struct
{
   float x;
   float y;
} Punt;
```

+

Funcions que operen amb Punt

```
void llegeixPunt(Punt& p);
void mostraPunt(Punt & p);
float distanciaPunts(Punt &p1, Punt &p2);
```

Classe Punt (dades + operacions)

```
class Punt
{
public:
   void llegeix();
   void mostra();
   float distancia(Punt &p);

   float m_x, m_y;
};
```

Una **classe** és una construcció que agrupa una estructura de dades amb les funcions que la manipulen

Membres d'una classe: atributs i mètodes

```
class Punt
{
public:
    void llegeix();
    void mostra();
    float distancia (Punt &p);

    float m_x, m_y;
};
```

Membres: Atributs + Mètodes

Mètodes (funcions)

Atributs (dades)

Les dades de la classe s'anomenen **atributs**.

- Els atributs representen les característiques/propietats de tots els objectes de la classe (en aquest cas, tots els punts tenen dues coordinades x i y)

Les funcions de la classe s'anomenen **mètodes**.

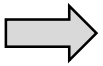
- Els mètodes representen les accions que podem fer amb els objectes de la classe (en aquest cas, un punt es pot llegir, es pot mostrar per pantalla i es pot calcular la seva distància a un altre punt)

Els atributs i mètodes de la classe són els **membres** de la classe

Objectes: instàncies (variables) d'una classe

Classe Punt:

```
class Punt
{
public:
  void llegeix();
  void mostra();
  float distancia(Punt &p);

  float m_x, m_y;
};
```



Objectes p1, p2

```
Punt p1;
p1.m_x = 0.0;
p1.m_y = 0.0;
P1.mostra()


Punt p2;
p2.llegeix();
p2.mostra();
```

- Les variables d'una classe s'anomenen **objectes.** En aquest cas, **p1** i **p2** són objectes de la classe **Punt**.
- Els objectes són instàncies diferents d'una classe. Cada objecte tindrà uns valors diferents pels atributs (propietats) de la classe
- Els objectes contenen tots els membres de la classe. No només les dades (atributs), sinó també les funcions (mètodes).
- L'accés als membres d'una classe (tant atributs com mètodes) es fa de forma similar a l'accés als camps d'un registre:

  `<objecte>.<membre>`

Objectes: crida als mètodes

Registre Punt

```
typedef struct
{
   float x;
   float y;
} Punt;
```

```
void llegeixPunt(Punt& p);
void mostraPunt(Punt & p);
float distanciaPunts(Punt &p1, Punt p2);
```

```
Punt p;
llegeixPunt(p);
mostraPunt(p);
```

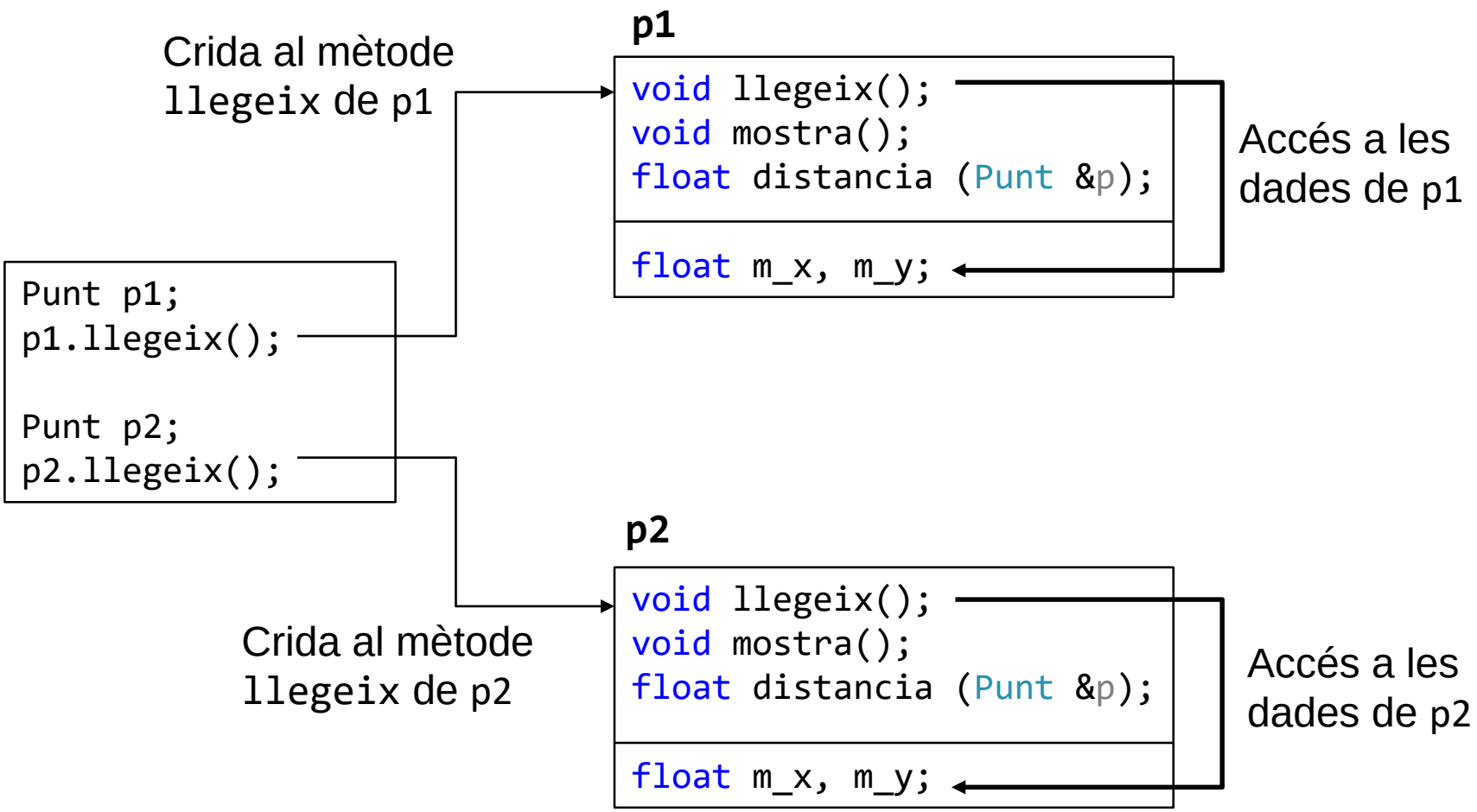
Classe Punt :

```
class Punt
{
public:
   void llegeix();
   void mostra();
   float distancia (Punt &p);

   float m_x, m_y;
};
```

```
Punt p;
p.llegeix();
p.mostra();
```

- La crida als mètodes es fa a partir d'un objecte específic de la classe
- Els mètodes són funcions (amb paràmetres i valor de retorn com les funcions habituals) que tenen accés a les dades (atributs) de l'objecte que utilitzem per fer la crida
- L'objecte que utilitzem per fer la crida desapareix de la capçalera de la funció. Es converteix en un paràmetre implícit de la crida al mètode

Objectes: crida als mètodes



Exercici 1

Torna a escriure aquest programa substituint la utilització de l'`struct Punt` per la `classe Punt`

```
int main()
{
   Punt p1 = { 0, 0 };
   Punt p2;

   llegeixPunt(p2);

   float distancia = distanciaPunts(p1, p2);

   cout << "Primer punt: ";
   mostraPunt(p1);
   cout << endl;
   cout << "Segon punt: ";
   mostraPunt(p2);
   cout << endl;
   cout << "Distancia: " << distancia <<
         endl;

    return 0;
}
```

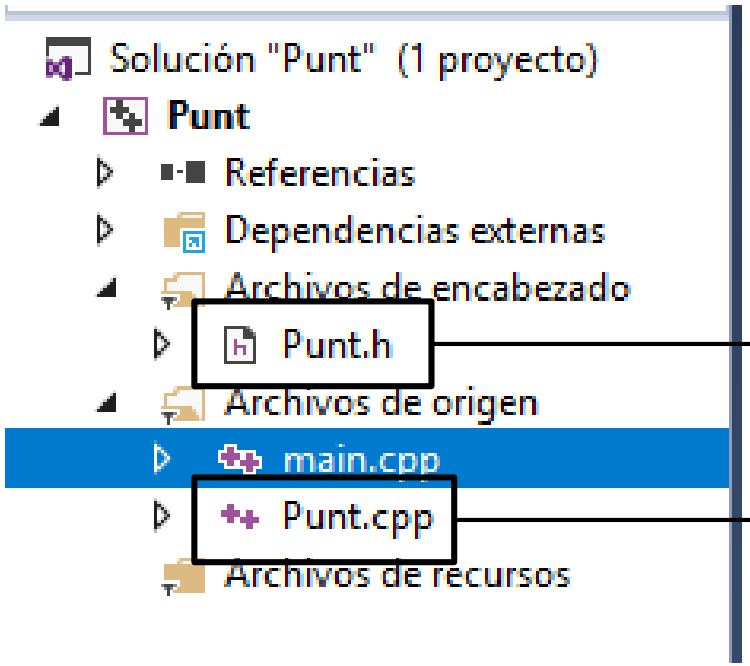
```
typedef struct
{
   float x;
   float y;
} Punt;
```

```
class Punt
{
public:
   void llegeix();
   void mostra();
   float distancia (Punt &p);

   float m_x, m_y;
};
```

Exercici 1: solució

```
int main()
{
   Punt p1;
   p1.m_x = 0.0;
   p1.m_y = 0.0;

   Punt p2;
   p2.llegeix();

   float distancia = p1.distancia(p2);

   cout << "Primer punt: ";
   p1.mostra();
   cout << endl;
   cout << "Segon punt: ";
   p2.mostra();
   cout << endl;
   cout << "Distancia: " << distancia << endl;

   return 0;
}
```

Classes: implementació



Declaració de la classe

- Fitxer "`Punt.h`"
- Declaració d'atributs i mètodes de la classe

Definició de la classe

- Fitxer "`Punt.cpp`"
- Implementació dels mètodes de la classe

Classes: implementació

Declaració de la classe: fitxer "Punt.h"

```
class Punt
{
public:
  void llegeix();
  void mostra();
  float distancia (Punt &p);

  float m_x, m_y;
};
```

Classes: implementació

Definició de la classe: fitxer "Punt.cpp"

```
#include "Punt.h"
```

Inclusió del fitxer de capçalera amb declaració de la classe

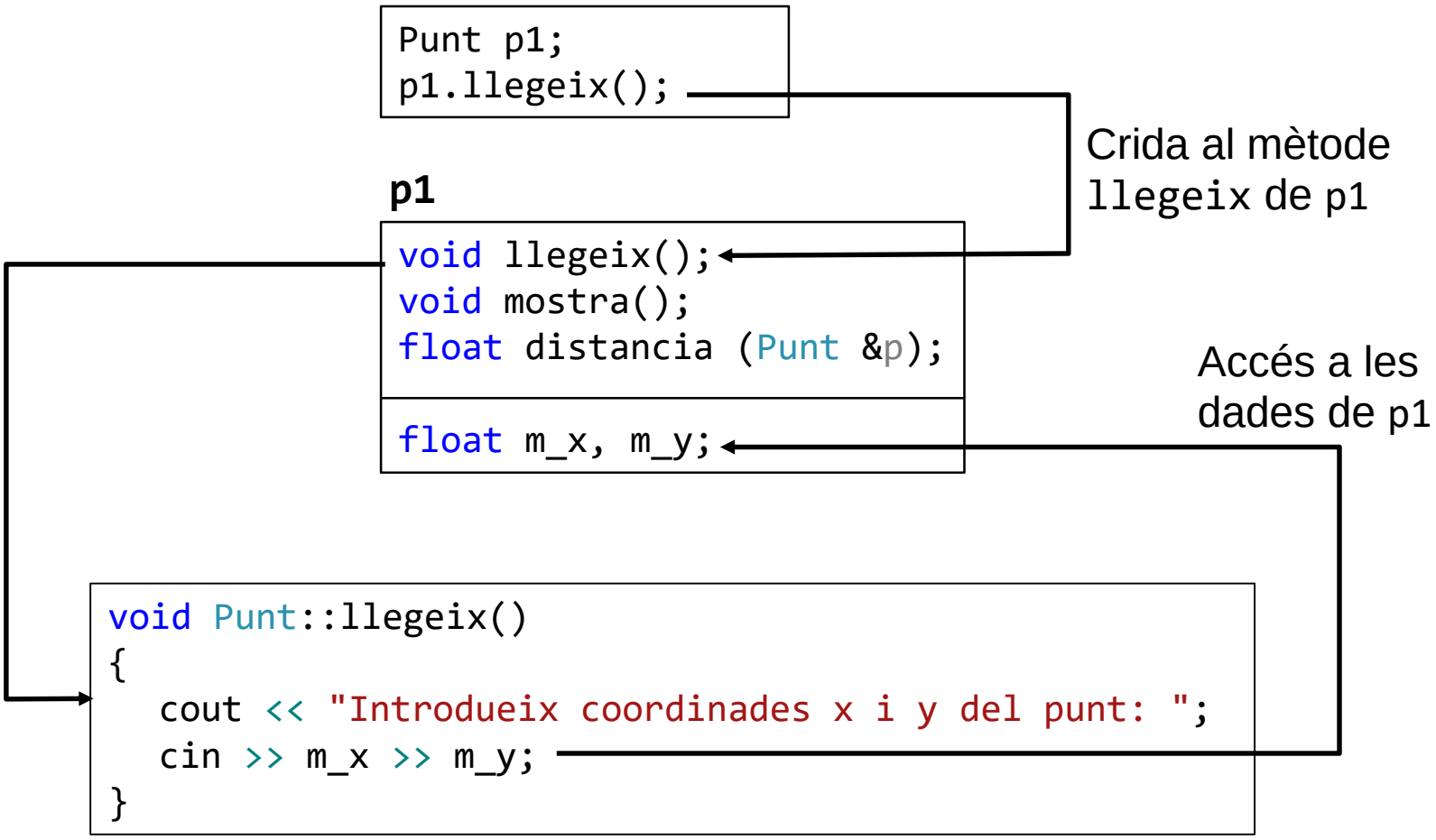
```
void Punt::llegeix()
{
   cout << "Introdueix coordinades x i y del punt: ";
   cin >> m_x >> m_y;
}

void Punt::mostra()
{
   cout << "(" << m_x << ", " << m_y << "
}
```

```
class Punt
{
public:
   void llegeix();
   void mostra();
   float distancia (Punt &p);

   float m_x, m_y;
};
```

Indicador que la funció és un mètode de la classe Punt

Accés als membres de la classe

Objectes: crida als mètodes



Exercici 2

- Completeu la definició de la classe `Punt` fent la implementació del mètode `distancia`.
- Afegiu un nou mètode `suma` a la classe `Punt` que rebi com a paràmetre un objecte de la classe `Punt` i retorni un nou punt en què les seves coordinades siguin la suma de les coordinades del punt que s'utilitza al fer la crida i les coordinades del punt que es passa com a paràmetre.

  Afegiu el nou mètode tant a la declaració (fitxer `Punt.h`) com a la definició (fitxer `Punt.cpp`) de la classe.

```cpp
#include "Punt.h"

void Punt::llegeix()
{
   cout << "Introdueix coordinades x i y del punt: ";
   cin >> m_x >> m_y;
}

void Punt::mostra()
{
   cout << "(" << m_x << ", " << m_y << ")";
}
```

```cpp
class Punt
{
public:
   void llegeix();
   void mostra();
   float distancia (Punt &p);

   float m_x, m_y;
};
```

Exercici 2: solució

```
class Punt
{
public:
   void llegeix();
   void mostra();
   float distancia(Punt& p);
   Punt suma(Punt& p);

   float m_x, m_y;
};
```

```
float Punt::distancia(Punt& p)
{
   float dx = m_x - p.m_x;
   float dy = m_y - p.m_y;
   return sqrt(dx*dx + dy * dy);
}

Punt Punt::suma(Punt& p)
{
   Punt pSuma;
   pSuma.m_x = m_x + p.m_x;
   pSuma.m_y = m_y + p.m_y;
   return pSuma;
}
```

Exercici 3: voluntari pujar nota

Implementeu una calculadora de nombres complexos seguint els passos següents:

Definiu una classe `Complex` amb dos atributs per guardar la part real (m_real) i la part imaginària (m_img) d'un nombre complex i els mètodes següents:

- Mostrar el nombre complex per pantalla en format `a + bi`
- Sumar dos nombres complexos
- Restar dos nombres complexos
- Multiplicar dos nombres complexos
- Avaluar el tipus d'operació a realitzar

El programa principal NO s'ha de modificar (s'ha implementat per l'avaluació de l'exercici). Aquest va cridant als diferents mètodes d'operacions (suma, resta, multiplicació). En cadascuna hi ha uns paràmetres per programa i es mostrarà el resultat final per pantalla.
Teniu un codi inicial a l'enllaç del caronte, a partir del qual haureu de treballar.

Exercici 3: solució

```
class Complex
{
public:
     float getReal(); //TO VALIDATE TEST
     float getImg(); //TO VALIDATE TEST
     void setReal(float pReal); //TO VALIDATE TEST
     void setImg(float PImg); //TO VALIDATE TEST
     void mostra() const;
     Complex suma(const Complex &c) const;
     Complex resta(const Complex &c) const;
     Complex multiplica(const Complex &c) const;
     Complex avaluaOperacio(const char operacio, const Complex& c2) const;

     float m_real, m_img;
};
```

Exercici 3: solució

```
#include "complex.h"

void Complex::mostra()
{
   cout << m_real << "+" << m_img << "i";
}

Complex Complex::suma(Complex& c)
{
   Complex resultat;

   resultat.m_real = m_real + c.m_real;
   resultat.m_img = m_img + c.m_img;
   return resultat;
}
```

Exercici 3: solució

```
Complex Complex::resta(Complex &c)
{
   Complex resultat;

   resultat.m_real = m_real - c.m_real;
   resultat.m_img = m_img - c.m_img;
   return resultat;
}

Complex Complex::multiplica(Complex &c)
{
   Complex resultat;

   resultat.m_real = (m_real*c.m_real) - (m_img*c.m_img);
   resultat.m_img = (m_real*c.m_img) + (m_img*c.m_real);
   return resultat;
}
```

Exercici 3: solució

```
Complex Complex::avaluaOperacio(const char operacio, const Complex& c2) const
{
    Complex resultat;

    switch (operacio)
    {
        case '1':
            resultat = suma(c2);
            break;
        case '2':
            resultat = resta(c2);
            break;
        case '3':
            resultat = multiplica(c2);
            break;
    }
    return resultat;
};
```