MP 20-21 Tema 1
Encapsulament de dades
Sessió 6: Programació orientada a objectes

Encapsulament de dades

Exemple

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

   Punt p2;
   p2.llegeix();
   if ((p2.m_x != 0) && (p2.m_y != 0))
   {
      float distancia = p1.distancia(p2);

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

   return 0;
}
```

???

???

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

   float m_x, m_y;
};
```

Què passa si volem canviar la representació de les dades del punt?

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

   float m_coord[2];
};
```

Encapsulament de dades

Exemple

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

???

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

   float m_coord[2];
};
```

- Els clients (o usuaris) d'una classe no han de tenir accés directe a la representació interna (propietats o atributs) de la classe.
- Qualsevol canvi o consulta a l'estat intern de la classe s'ha de fer utilitzant els mètodes (o accions) definides sobre la classe.

Distingir entre part privada i part pública de la classe

Encapsulament de dades: `private` i `public`

Part pública

- Accessible des de fora de la classe (mètodes d'altres classes i funcions globals) i també pels mètodes de la pròpia classe
- Normalment conté només mètodes
- Defineix la interfície de la classe: conjunt d'accions o operacions que podem fer amb els objectes de la classe

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

private:
  float m_x, m_y;
};
```

Part privada

- Només és accessible dins dels mètodes de la pròpia classe
- No és visible ni accessible fora de la classe (mètodes d'altres classes i funcions globals)
- Normalment conté tots els atributs i també els mètodes que només es criden des d'altres mètodes de la classe

Encapsulament de dades: private i public

UAB

Exemple

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

   Punt p2;
   p2.llegeix();
   if ((p2.m_x != 0) && (p2.m_y != 0))
   {
      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;
}
```

Com podem modificar o recuperar el valor dels atributs privats de la classe?

```
class Punt
{
public:
   void llegeix();
   void mostra();
   float distancia (Punt &p);
private:
   float m_x, m_y;
};
```

Encapsulament de dades: getters i setters

```
class Punt
{
public:
   float getX();
   float getY();
   void setX(float x);
   void setY(float y);
   void llegeix();
   void mostra();
   float distancia (Punt &p);
private:
   float m_x, m_y;
};
```

`getters`: mètodes per recuperar el valor dels atributs

`setters`: mètodes per modificar el valor dels atributs

Apart de recuperar/modificar el valor dels atributs poden incloure altres accions per controlar errors, validar el rang dels valors, etc.

```
float Punt::getX()
{
   return m_x;
}

float Punt::getY()
{
   return m_y;
}
```

```
void Punt::setX(float x)
{
   m_x = x;
}

void Punt::setY(float y)
{
   m_y = y;
}
```

Encapsulament de dades: getters i setters

```
int main()
{
   Punt p1;
   p1.setX(0.0);
   p1.setY(0.0);
   Punt p2;
   p2.llegeix();

   if ((p2.getX() != 0) && (p2.getY() != 0))
   {
      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;
}
```

```
class Punt
{
public:
   float getX();
   float getY();
   void setX(float x);
   void setY(float y);
   void llegeix();
   void mostra();
   float distancia (Punt &p);
private:
   float m_x, m_y;
};
```

Exercici

A partir de la classe Punt que us donem ja feta:

- Feu tots els canvis que hem explicat anteriorment per separar entre part privada i part pública de la classe, introduint també els `getters` i `setters`.
- Feu tots els canvis necessaris a la classe Punt per poder canviar la representació interna del punt tal com s'indica

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

   Punt p2;
   p2.llegeix();
   if ((p2.m_x != 0) && (p2.m_y != 0))
   {
      float distancia = p1.distancia(p2);

     ...
   }
   return 0;
}
```

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

   float m_x, m_y;
};
```



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

   float m_coord[2];
};
```

Exercici: solució

```
class Punt
{
public:
   float getX();
   float getY();
   void setX(float x);
   void setY(float y);
   void llegeix();
   void mostra();
   float distancia (Punt &p);
private:
   float m_coord[2];
};
```

```
void Punt::llegeix()
{
   cin >> m_coord[0] >> m_coord[1];
}
void Punt::mostra()
{
   cout << "(" << m_coord[0] << ", " <<
     m_coord[1] << ")";
}
float Punt::distancia(Punt &p)
{
   float dx = m_coord[0] - p. m_coord[0];
   float dy = m_coord[1] - p. m_coord[1];
   return sqrt(dx*dx + dy * dy);
}
```

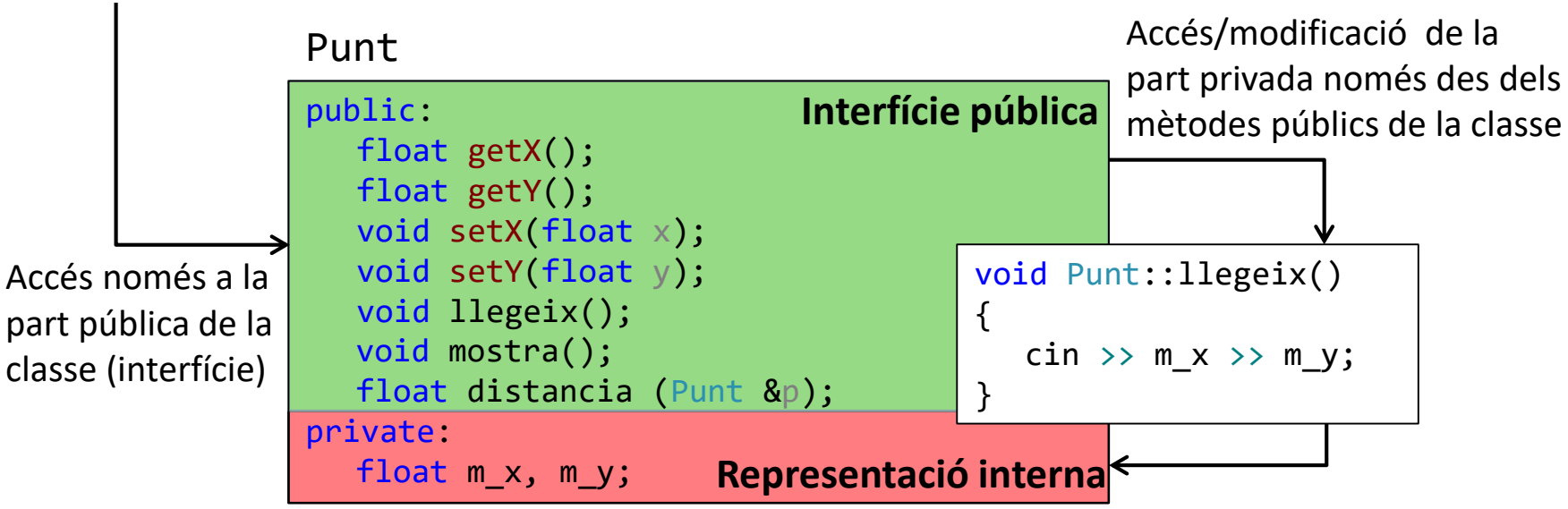
```
float Punt::getX()
{
   return m_coord[0];
}
float Punt::getY()
{
   return m_coord[1];
}
```

```
void Punt::setX(float x)
{
   m_coord[0] = x;
}
void Punt::setY(float y)
{
    m_coord[1] = y;
}
```

Abstracció

- Definició d'una interfície pública amb les accions i/o operacions (mètodes) que es poden utilitzar per interactuar amb els objectes de la classe.
- La interfície pública amaga (abstrau) la representació interna de la classe als programes/classes (clients) que la utilitzen.
- Els clients no necessiten conèixer la representació interna de la classe per poder-la utilitzar.

```
Punt p;
p.llegeix();
p.setX(1.0);
p.mostra();
```



Classes: mètodes *inline*

```
class Punt
{
public:
   float getX();
   float getY();
   void setX(float x);
   void setY(float y);
   void llegeix();
   void mostra();
   float distancia (Punt &p);
private:
   float m_x, m_y;
};
```

```
class Punt
{
public:
   float getX() { return m_x; }
   float getY() { return m_y; }
   void setX(float x) { m_x = x; }
   void setY(float y) { m_y = y; }
   void llegeix();
   void mostra();
   float distancia(Punt &p);
private:
   float m_x, m_y;
};
```

```
float Punt::getX()
{
   return m_x;
}

float Punt::getY()
{
   return m_y;
}
```

Mètodes inline:

- Implementació dins de la declaració de la classe
- S'utilitza per implementar mètodes simples i que s'utilitzen amb freqüència.
- No es fa crida a la funció. Internament el compilador substitueix la crida pel codi de la funció

Exercici voluntari pujar nota

Implementeu una classe anomenada `Recta`, que permeti representar una recta i fer algunes operacions entre rectes i entre rectes i punts. Aquesta classe ha de tenir:

- Els atributs necessaris per guardar els paràmetres de la recta utilitzant l'equació general de la recta. L'equació general de la recta representa la recta amb l'expressió $Ax + By + C = 0$, on A, B i C són els paràmetres que defineixen la recta.
- Mètodes per modificar i recuperar cadascun dels paràmetres de la recta: `setA`, `setB`, `setC` i `getA`, `getB`, `getC`.
- Un mètode que calculi la distància entre la recta i un punt:

```
float distancia(Punt &pt);
```

La distància d'un punt a la recta es pot calcular amb l'expressió següent:

$$d = \frac{|Ax + By + C|}{\sqrt{A^2 + B^2}}$$

Exercici voluntari pujar nota

- Un mètode que calculi el punt d'intersecció entre dues rectes:

```
bool intersecció(Recta& r, Punt &pt);
```

Aquest mètode ha de retornar `true` si existeix un punt d'intersecció entre les dues rectes o `false` si són paral·leles i no hi ha punt d'intersecció. Si existeix, el punt d'intersecció s'ha de retornar utilitzant el paràmetre per referència de la classe `Punt`.

El punt d'intersecció de dues rectes expressades amb l'equació general de la recta, r_1: $A_1x + B_1y + C_1 = 0$ i r_2: $A_2x + B_2y + C_2 = 0$ es pot calcular d'aquesta forma:

$$x = \frac{B_1C_2 - B_2C_1}{B_2A_1 - B_1A_2} \qquad y = \frac{A_2C_1 - A_1C_2}{B_2A_1 - B_1A_2}$$

Nota: utilitzeu la classe `Punt` que heu implementat a l'exercici anterior.

Exercici: solució

1. Implementeu una classe anomenada `Recta`, que permeti representar una recta i fer algunes operacions entre rectes i entre rectes i punts.

```
class Recta
{
public:
  void setA(float a) { m_a = a; }
  void setB(float b) { m_b = b; }
  void setC(float c) { m_c = c; }
  float getA() { return m_a; }
  float getB() { return m_b; }
  float getC() { return m_c; }
  float distancia(Punt &pt);
  bool intersecció(Recta& r, Punt &pt);
private:
  float m_a;
  float m_b;
  float m_c;
};
```

Exercici: solució

- Un mètode que calculi la distància entre la recta i un punt:

```
float distancia(Punt &pt);
```

La distància d'un punt a la recta es pot calcular amb l'expressió següent:

$$d = \frac{|Ax + By + C|}{\sqrt{A^2 + B^2}}$$

```
float Recta::distancia(Punt &pt)
{
  return abs(m_a * pt.getX() + m_b * pt.getY() + m_c) /
                sqrt((m_a * m_a) + (m_b * m_b));
}
```

Exercici: solució

- Un mètode que calculi el punt d'intersecció entre dues rectes:

```
bool Recta::interseccio(Recta &r, Punt& pt)
{
   bool interseccio = true;
   float denominador = (r.m_b * m_a) - (m_b * r.m_a);
   if (abs(denominador) < 0.0001)
      interseccio = false;
   else
   {
      pt.setX(((m_b * r.m_c) - (r.m_b * m_c)) / denominador);
      pt.setY(((r.m_a * m_c) - (m_a * r.m_c)) / denominador);
   }
   return interseccio;
}
```