ESTRUCTURAS DE DATOS

NOTAS SOBRE C++

Objetos y memoria dinámica

Manuel Montenegro Montes
Departamento de Sistemas Informáticos y Computación
Facultad de Informática – Universidad Complutense de Madrid

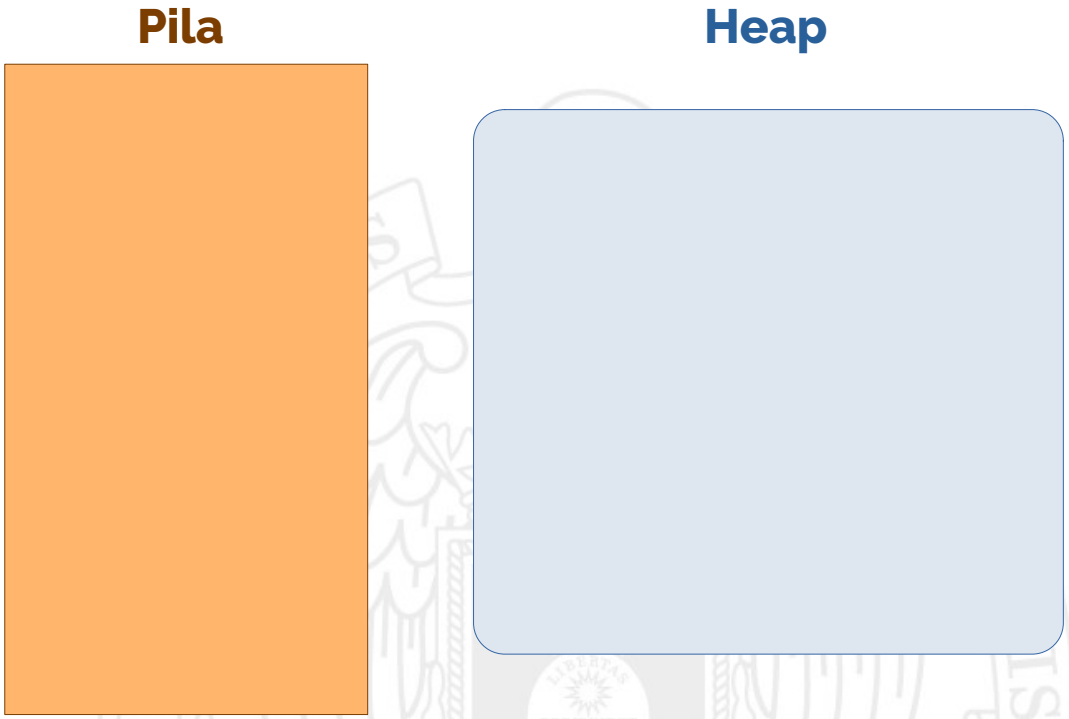
Regiones de memoria: pila y *heap*

Regiones de memoria

- **Memoria principal (global)**: variables globales.
  - Se reserva al iniciarse el programa, y se libera al finalizarse.
- **Pila**: variables locales, parámetros.
  - Se reserva y libera a medida que estas variables entran en ámbito y salen de ámbito, respectivamente.
- ***Heap***: memoria dinámica.
  - Se reserva y libera manualmente mediante `new` y `delete`.
  - Solamente es accesible a través de punteros.

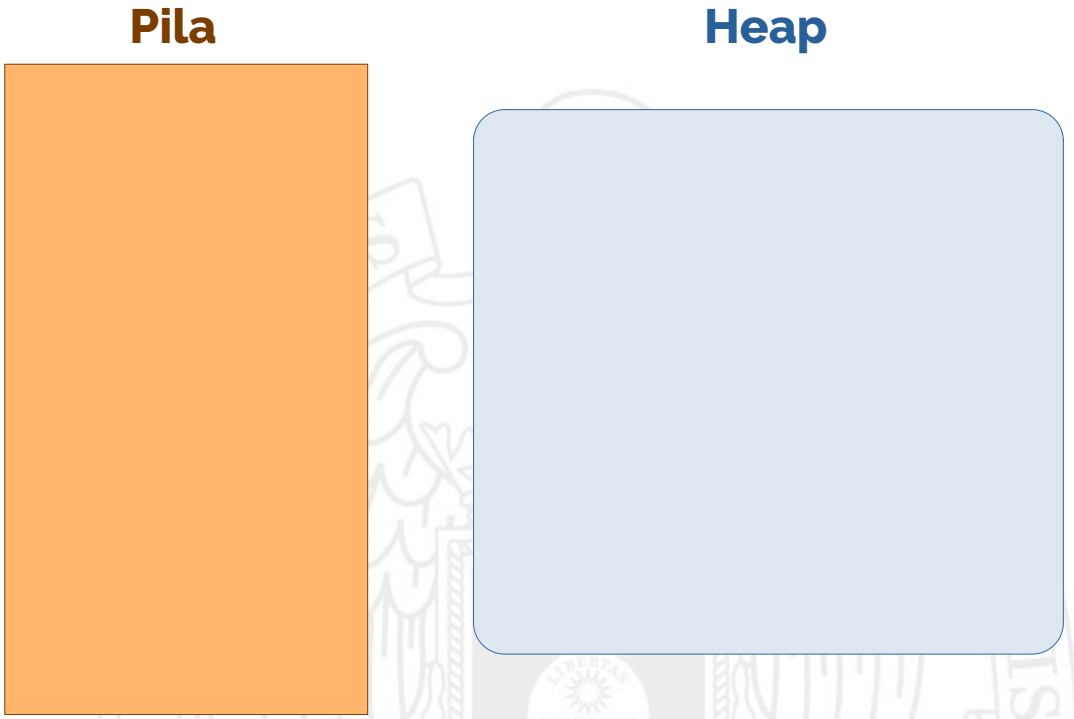
Regiones de memoria

```
int main() {
  int x = 3;
  int *y = new int;
  *y = 3;
  int *z = &x;

  delete y;
  return 0;
}
```



Regiones de memoria

```
int main() {
  int *xs = new int[4];
  xs[0] = 3;
  xs[1] = 7;

  int ys[3];

  delete[] xs;
  return 0;
}
```



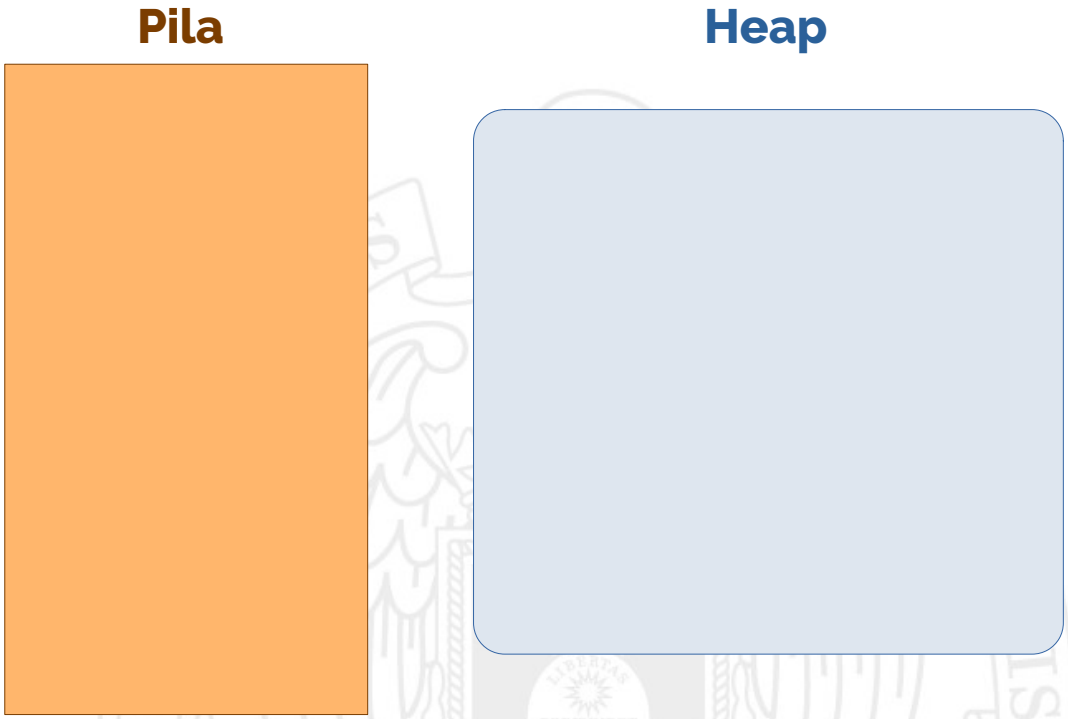
Creación de objetos en el *heap*

Recordatorio: clase Fecha

```
class Fecha {
public:
  Fecha(int dia, int mes, int anyo);
  Fecha(int anyo);

  int get_dia() const;
  void set_dia(int dia);
  int get_mes() const;
  void set_mes(int mes);
  int get_anyo() const;
  void set_anyo(int anyo);
  void imprimir();

private:
  int dia;
  int mes;
  int anyo;
};
```

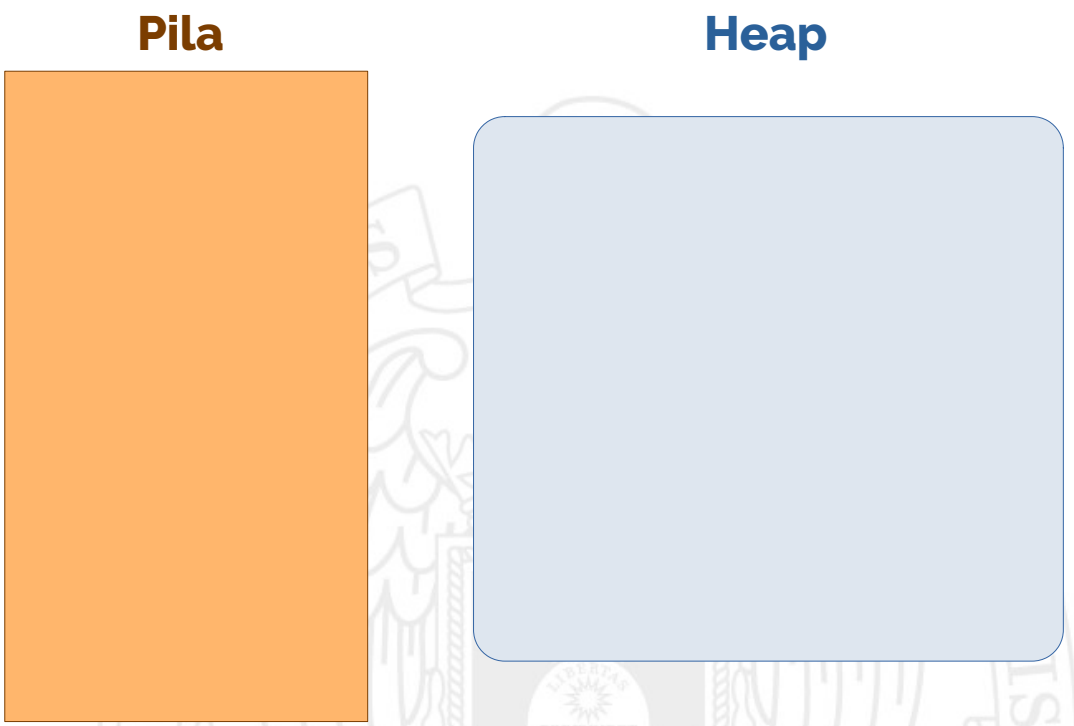
Creación de instancias en la pila y *heap*

```
int main() {
  Fecha f1(28, 8, 2038);
  Fecha *f2 = new Fecha(10, 6, 2010);

  std::cout << "Fecha 1: ";
  f1.imprimir();
  std::cout << std::endl;

  std::cout << "Fecha 2: ";
  f2→imprimir();
  std::cout << std::endl;

  delete f2;
  return 0;
}
```



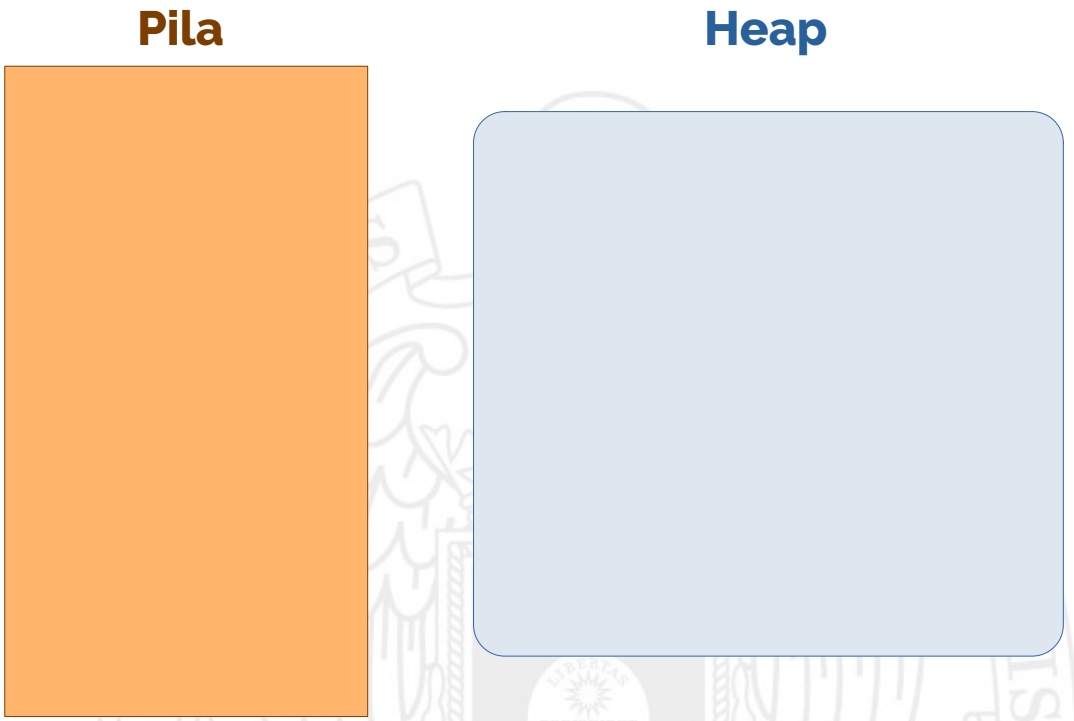
Arrays de objetos

```
int main() {
  Fecha fs[3] =
     { {2010}, {2011}, {2012} };

  return 0;
}
```



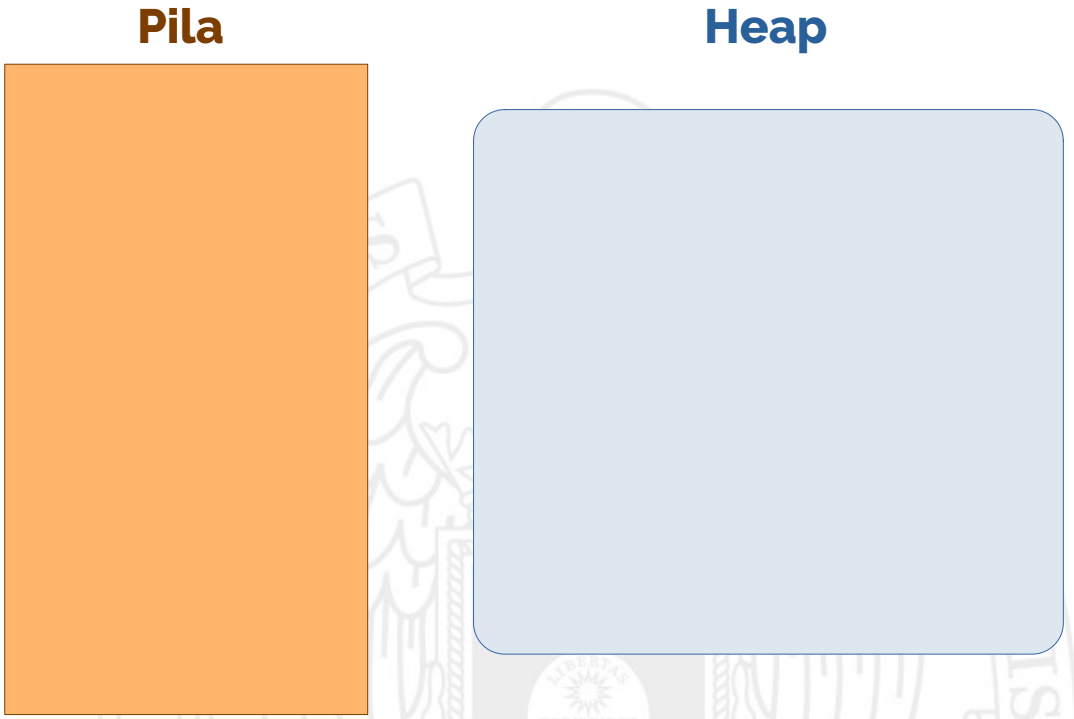
Arrays de punteros a objetos

```
int main() {
  Fecha *fs[3];
  fs[0] = new Fecha(2010);
  fs[1] = new Fecha(2011);
  fs[2] = new Fecha(2012);

  delete fs[0];
  delete fs[1];
  delete fs[2];

  return 0;
}
```



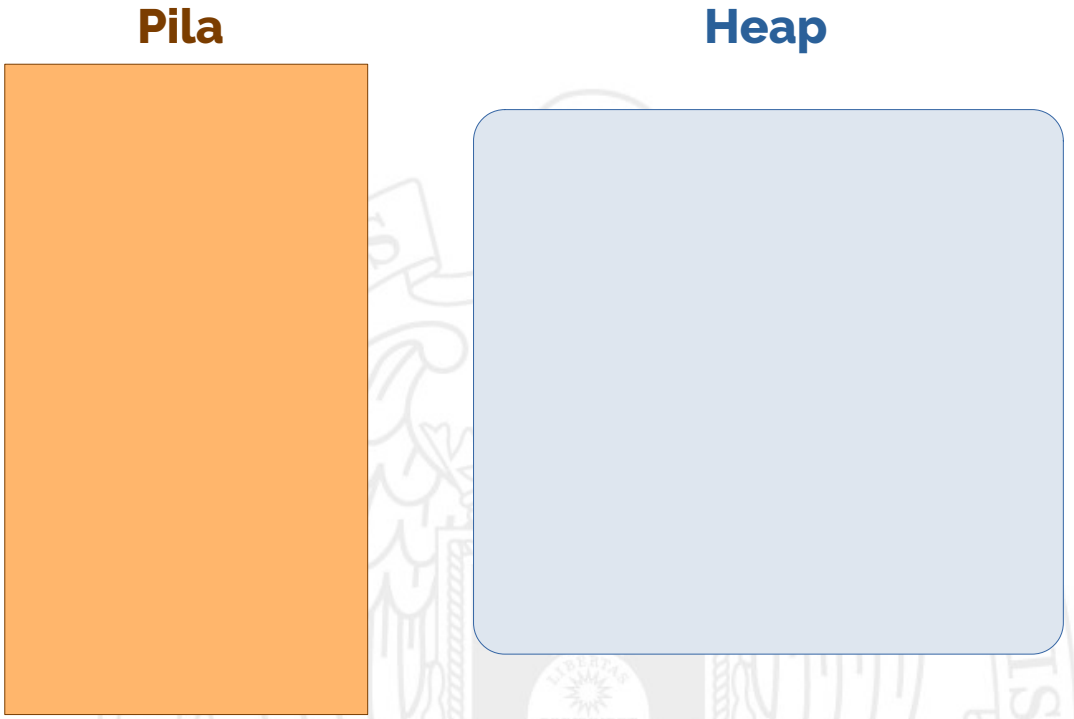
Arrays dinámicos de punteros a objetos

```
int main() {
  Fecha **fs = new Fecha*[3];
  fs[0] = new Fecha(2010);
  fs[1] = new Fecha(2011);
  fs[2] = new Fecha(2012);

  delete fs[0];
  delete fs[1];
  delete fs[2];
  delete[] fs;

  return 0;
}
```



Arrays dinámicos de objetos

```
int main() {
  Fecha *fs = new Fecha[3];

  delete[] fs;
  return 0;
}
```

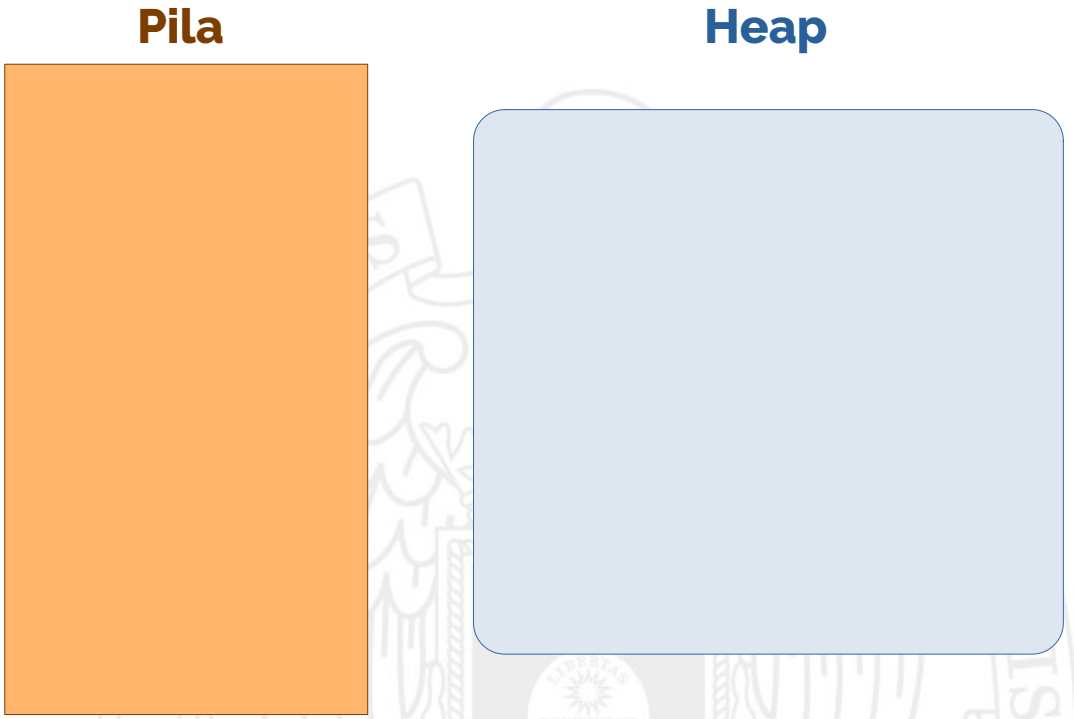


Añadiendo un constructor por defecto

```
class Fecha {
public:
  Fecha(int dia, int mes, int anyo);
  Fecha(int anyo);
  Fecha(): Fecha(1, 1, 1900) { }

  int get_dia() const;
  void set_dia(int dia);
  int get_mes() const;
  void set_mes(int mes);
  int get_anyo() const;
  void set_anyo(int anyo);
  void imprimir();

private:
  int dia;
  int mes;
  int anyo;
};
```

Arrays dinámicos de objetos

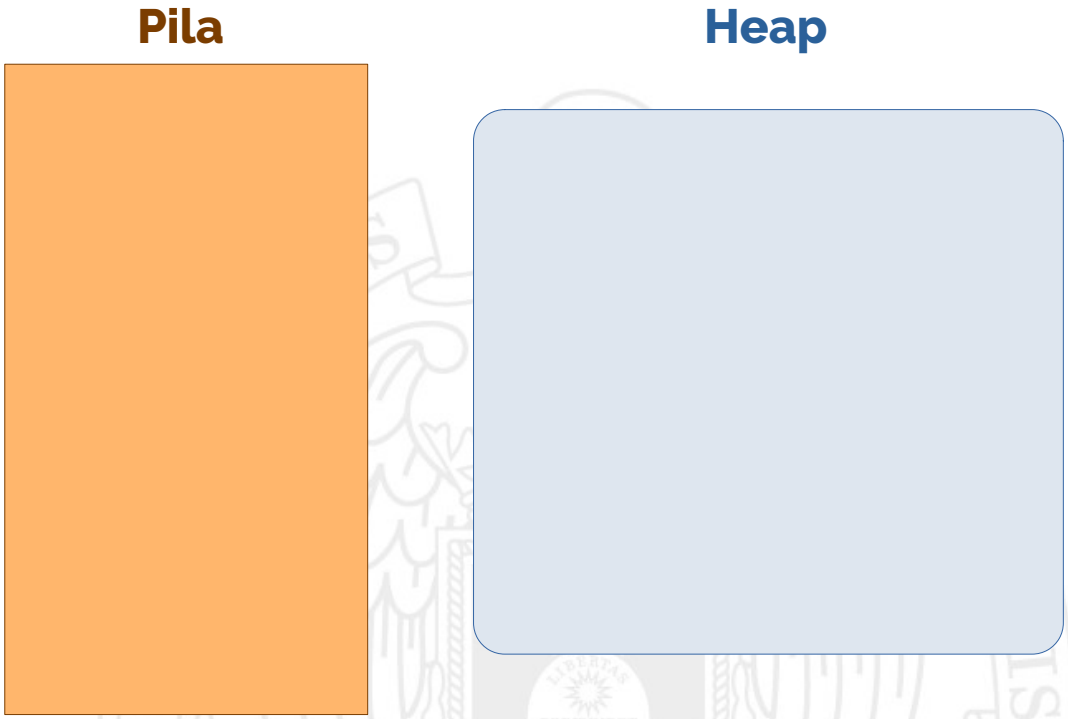
```
int main() {
  Fecha *fs = new Fecha[3];

  delete[] fs;
  return 0;
}
```



Compartición de objetos

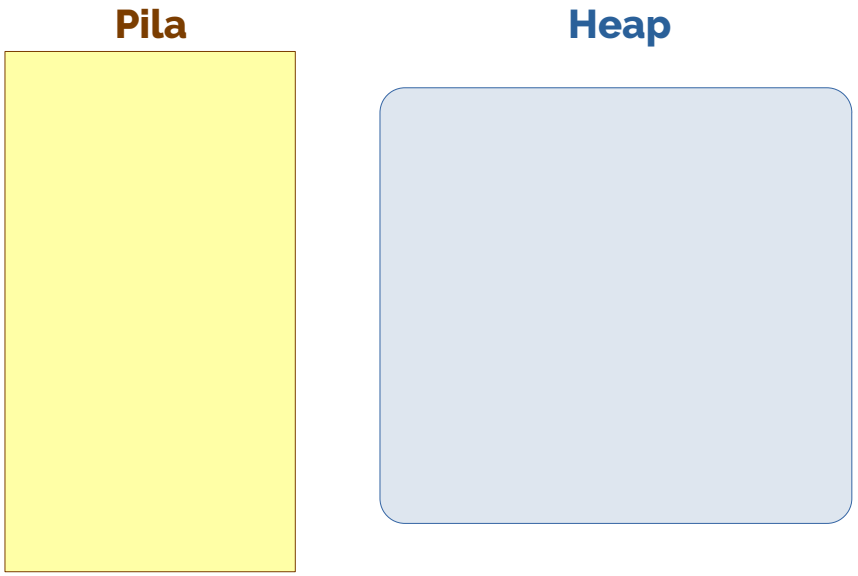
Compartición de punteros

```
int main() {
  Fecha *f1 = new Fecha(28, 8, 2019);
  Fecha *f2 = f1;

  f1->imprimir();
  f2->imprimir();

  f1->set_dia(1);

  f1->imprimir();
  f2->imprimir();

  delete f1;
  // delete f2
  return 0;
}
```

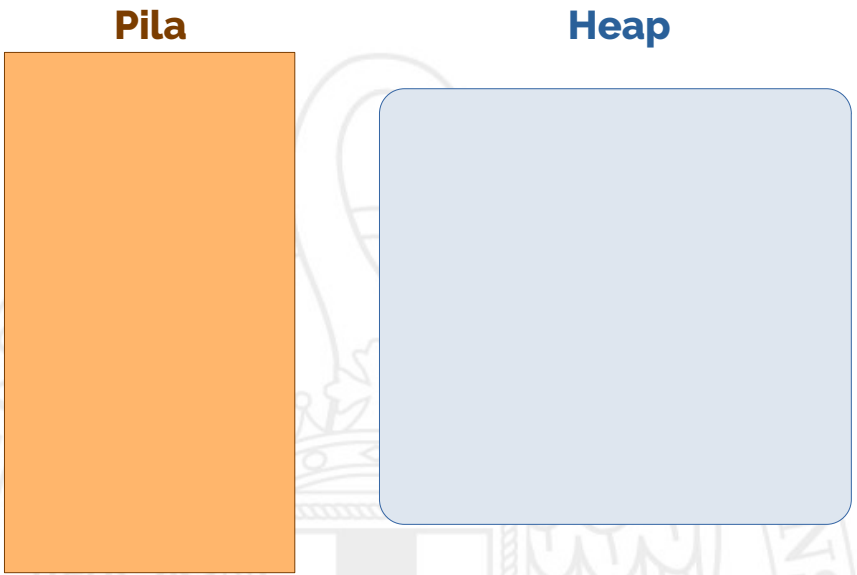


Comparación con Java

Java vs C++



- **Todos los objetos viven en el heap.**
- La pila solo almacena valores básicos o punteros a objetos.



- **Los objetos pueden almacenarse en el heap o en la pila.**

Manejo de objetos en Java y C++

```java
public static void main(String[] args) {
  Fecha f = new Fecha(20, 3, 2010);
  f.imprimir();
}
```

En Java tenemos que crear el objeto `f` en el *heap*, porque todos los objetos se crean allí.

```cpp
int main() {
  Fecha *f = new Fecha(20, 3, 2010);
  f→imprimir();

  delete f;
  return 0;
}
```

En C++ no es necesario crear el objeto en el *heap*.

Manejo de objetos en Java y C++

```java
public static void main(String[] args) {
  Fecha f = new Fecha(20, 3, 2010);
  f.imprimir();
}
```

En Java tenemos que crear el objeto `f` en el *heap*, porque todos los objetos se crean allí.

```cpp
int main() {
  Fecha f(20, 3, 2010);
  f.imprimir();

  return 0;
}
```

En C++ es más sencillo crear el objeto `f` en la pila.

¿Cuándo se utiliza el heap en C++?

Lo vamos a utilizar en estas situaciones:

- Cuando el tamaño de un array no es conocido en tiempo de compilación.
- Para estructuras de datos recursivas.
  - Por ejemplo, nodos de árboles y listas enlazadas.