ESTRUCTURAS DE DATOS

APLICACIONES DE TIPOS ABSTRACTOS DE DATOS

# Gestión de una academia (3)

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

# Lista de espera

# Requisitos

- Cada curso tiene una lista de espera.
- Si un estudiante se matricula en un curso y no hay plazas disponibles, se le pone en lista de espera.
- Cuando un estudiante se da de baja en un curso, se matricula automáticamente al primero de la lista de espera (si existe).
- Operaciones añadidas o modificadas:
  - Dar de baja a un estudiante.
  - La operación de matrícula de un curso devuelve un booleano indicando si el estudiante ha sido matriculado o está en lista de espera.

# Lista de espera en cursos

```
class Academia {
public:
  …

private:
  struct InfoCurso {
    std::string nombre;
    int numero_plazas;
    std::unordered_set<Estudiante> estudiantes;
    std::queue<Estudiante> lista_espera;

    InfoCurso(const std::string &nombre,
              int numero_plazas);
  };

  std::unordered_map<Curso, InfoCurso> cursos;
}
```

# Matrícula en un curso (cambios)

```
class Academia {
public:
  bool matricular_en_curso(const Estudiante &est, const Curso &curso) {
    InfoCurso &info_curso = buscar_curso(curso);
    InfoEstudiante &info_est = buscar_estudiante(est);
    if (info_curso.estudiantes.contains(est)) {
      throw std::domain_error("estudiante ya matriculado");
    }

    if (info_curso.estudiantes.size() < info_curso.numero_plazas) {
      info_curso.estudiantes.insert(est);
      info_est.cursos.insert(curso);
      return true;
    } else {
      info_curso.lista_espera.push(est);
      return false;
    }
  }
  …
private:
  …
  std::unordered_map<Curso, InfoCurso> cursos;
}
```

# Darse de baja en un curso

```
class Academia {
public:
  void dar_de_baja_en_curso(const Estudiante &id_est, const Curso &nombre_curso) {
    InfoCurso &curso = buscar_curso(nombre_curso);

    auto it_estudiante = curso.estudiantes.find(id_est);
    if (it_estudiante ≠ curso.estudiantes.end()) {
      curso.estudiantes.erase(it_estudiante);
      it_estudiante→cursos.erase(curso.nombre);

      while (!curso.lista_espera.empty() && curso.estudiantes.size() < curso.numero_plazas) {
        const Estudiante &nif_primero = curso.lista_espera.front();
        curso.lista_espera.pop();
        if (!curso.estudiantes.contains(nif_primero)) {
          curso.estudiantes.insert(nif_primero);
          estudiantes.at(nif_primero).cursos.insert(curso.nombre);
        }
      }
    }
  }
  …
private:
  …
  std::unordered_map<Curso, InfoCurso> cursos;
}
```