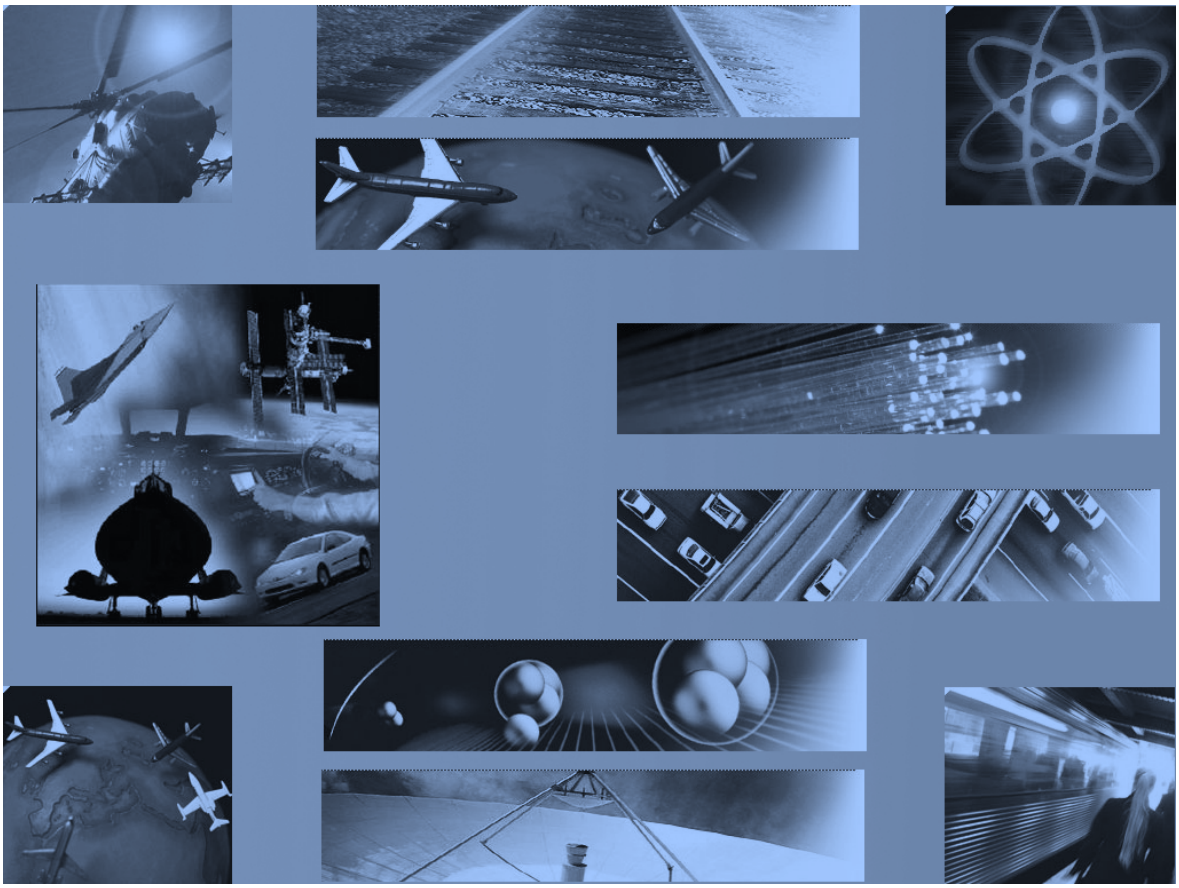
EJERCICIOS DE LA ASIGNATURA TIEMPO-REAL



GRADO DE INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA

EJERCICIOS DE DISEÑO DE CLASES

EJERCICIO 1

Diseñe e implemente una clase denominada **Sphere** que contenga datos de instancia que representen el diámetro de una esfera. Defina el constructor de **Sphere** para que acepte e inicialice el diámetro e incluya métodos de consulta y de configuración del parámetro. Incluya métodos que calculen y devuelvan el área superficial $(4 \cdot \pi \cdot r^2)$ y el volumen $\left(4 \cdot \pi \cdot r^3/_3\right)$ de la esfera. Incluya un método **operator<<** que devuelva una descripción de la esfera de una sola línea. Cree una función **main** que instancie y actualice varios objetos **Sphere**. Haga que la función **main** solicite también al usuario los valores de entrada para crear algún objeto.

Por ejemplo, para añadir **operator<<** a la clase **Sphere** haremos:

```
class Sphere {
   // TODO: añadir el resto código que define Sphere
   friend std::ostream& operator<< (std::ostream& os, Sphere& sp) {
      // TODO: enviar a 'os' los datos de 'sp': os << sp.diam << std::endl;
      return os;
   }
}
```

¿Cómo puede asegurar que el programa es robusto frente a la entrada de datos erróneos por teclado (por ejemplo, si el usuario introduce 'abc' como diámetro de la esfera)?

EXTENSION: Diseñe una clase **Point_3D**, que representa un punto en el espacio, junto con un método para calcular la distancia a otro punto que se pasa como parámetro. Amplíe la **Sphere** para que tenga un centro, y añada un método que permita comprobar si un punto se encuentra dentro o fuera de la esfera. ¿Cómo se podría comprobar que está justo sobre su superficie?

EJERCICIO 2

Diseñe e implemente una clase **Dog** que contenga datos de instancia que representen el nombre y la edad del perro. Defina el constructor **Dog** para que acepte e inicialice los datos de instancia. Incluya métodos de consulta y configuración del nombre y de la edad. Incluya un método para calcular y devolver la edad del perro en "años humanos" (siete veces la edad del perro). Incluya un método **operator<<** que devuelva una descripción del perro en una sola línea. Cree una función **main** que instancie y actualice varios objetos **Dog**. Haga que la función **main** solicite también al usuario los valores de entrada para crear algún objeto.

EJERCICIO 3

Diseñe e implemente una clase **Box** que contenga datos de instancia que representen la altura, la anchura y la profundidad de una caja, en metros. Incluya también una variable **bool** denominada 'llena' que

represente si la caja está llena o no. Defina el constructor de **Box** para que acepte e inicialice la altura, la anchura y la profundidad de la caja. Cada objeto **Box** recién creado representará una caja vacía. Incluya métodos de consulta para todos los datos de la instancia. Incluya un método **`insertar_objeto`**, que acepta como parámetro el volumen de un objeto que se inserta en el objeto **Box**. El método devuelve un valor **`bool`** que indica si se ha podido meter en el objeto **Box** (y, por tanto, se reduce el volumen de almacenamiento disponible en el objeto **Box**) o si no ha cabido. Añada también un método **`vaciar_caja`**. Incluya un método **`operator<<`** que devuelva una descripción de la caja en una sola línea. Cree una función **`main`** que instancie y actualice varios objetos **Box**. Haga que la función **`main`** solicite también al usuario los valores de entrada para crear algún objeto.

Pista: tendrá que crear un atributo para contabilizar el volumen disponible en el objeto **Box**, aunque su valor no se pase en el constructor, sino que es calculado a partir de los valores de las dimensiones (altura, la anchura y la profundidad).

EJERCICIO 4

Diseñe e implemente una clase **Book** que contenga datos de instancia correspondientes al título, al autor, a la editorial y a la fecha de publicación de un libro. Defina el constructor **Book** para que acepte e inicialice estos datos. Incluya métodos de consulta y configuración para todos los datos de la instancia. Incluya un método **`operator<<`** que devuelva una descripción del libro en múltiples líneas, una por dato del libro, convenientemente formateada. Cree una función **`main`** que instancie y actualice varios objetos **Book**. Haga que la función **`main`** solicite también al usuario los valores de entrada para crear algún objeto (ojo ahora con los strings; puede necesitar utilizar la función **`getline`**).

EJERCICIO 5

Diseñe una clase **`Secret_Message`** que almacena un texto y una clave secreta que permite mostrar dicho texto, y que incluya además un constructor, tres métodos relacionados con la gestión de la clave, un método **`operator<<`** para imprimir por consola y un método **`erase`** para borrar completamente el mensaje. El constructor acepta el texto y la clave (alfanumérica, sin restricciones en cuanto al formato (del estilo "*tiene que tener al menos un número y un carácter que no sea número ni letra*")). La gestión de la clave se realiza mediante los métodos **`lock`**, **`unlock`** e **`is_locked`**. El método **`is_locked`** devuelve un valor booleano que indica si el objeto está bloqueado o no. Este método se puede invocar siempre. El método **`unlock`** recibe como parámetro una clave y desbloquea el objeto si coincide con la clave original. Una vez desbloqueado, el objeto se puede imprimir por pantalla (mediante **`operator<<`**), volver a bloquear (mediante el método **`lock`** sin parámetros), cambiar la clave (mediante el método **`lock`** con un parámetro 'clave_nueva') o borrar su contenido para siempre (mediante **`erase`**). Cree una función **`main`** que instancie y actualice varios objetos **`Secret_Message.`** Haga que la función **`main`** solicite también al usuario los valores de entrada para crear algún objeto.

EXTENSIÓN: amplíe el programa para que haya restricciones sobre la clave (del estilo "*tiene que tener al menos un número y un carácter que no sea número ni letra*", consulte) y para que el método lock que cambia la clave tenga dos parámetros: 'clave_antigua' y 'clave_nueva', de forma que solo se cambie la clave en caso de que la 'clave_antigua' coincida con la clave almacenada en **`Secret_Message.`**

EJERCICIO 6

Diseñe una clase **`Card`** que modela las cartas de la baraja española. Utilice tipos enumerados para definir los números y palos de la baraja, y defina funciones para imprimirlos por consola (**`operator<<`**). Una vez cread, un objeto de tipo **`Card`** no puede modificarse, aunque tiene que definir métodos para acceder a sus atributos (número y palo) y un constructor adecuado. Cree una función **`main`** que instancie y actualice varios objetos **`Card.`** Haga que la función **`main`** solicite también al usuario los valores de entrada para crear algún objeto.

Cree adicionalmente una clase **`Brisca`**, que tiene como atributos el palo de la "muestra" (establecido en el constructor y de solo lectura). Añada además los siguientes métodos:

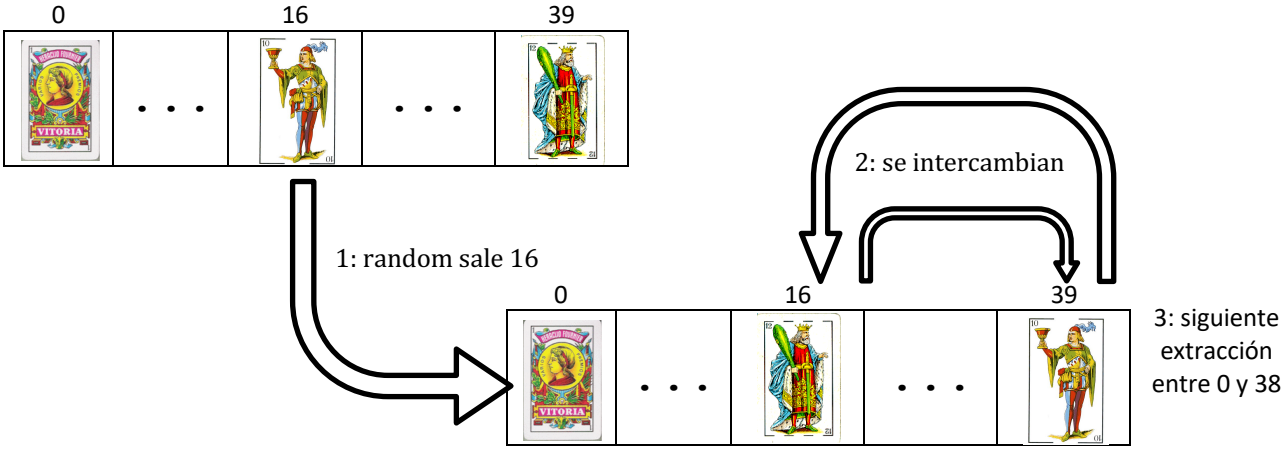
- **`puntuacion_carta`**: retorna el valor de la **`Card`** que se le pasa como parámetro según las reglas de la brisca (sota 2 puntos, caballo 3 puntos, rey 4 puntos, tres 10 puntos y as 11 puntos).
- **`gana_mano`**: recibe dos objetos de tipo **`Card`** (la carta que juega la persona que "va de mano" y la del contrincante) y devuelve true si gana el primero y false en caso contrario. Aplique las reglas de la brisca teniendo en cuenta el palo de la "muestra".

Amplíe la función **`main`** para comprobar que la clase **`Brisca`** funciona correctamente.

REPARTIR LAS CARTAS DE LA BARAJA ESPAÑOLA

El típico algoritmo que se desarrolla la primera vez que se enfrenta al problema de realizar "*un muestreo aleatorio sin sustitución*", como es el caso del reparto de las cartas de una baraja, es generar números aleatorios entre 1 y 40 para seleccionar una carta y luego llevar un conteo (generalmente en un vector) de qué cartas se han repartido hasta el momento. Si el número aleatorio generado coincide con el de una carta ya repartida, se genera uno nuevo hasta obtener uno que representa una carta no repartida. Este algoritmo tiene el problema de que cuando quedan pocas cartas por repartir se puede volver muy lento.

Un algoritmo que soluciona este problema consiste en generar un número aleatorio entre 0 y el total de cartas que quedan por repartir, e intercambiar la carta que acaba de ser elegida por la última carta del vector no elegida (la que ocupa la posición "total de cartas por repartir"). En la siguiente extracción se generará un número entre 0 y tamaño-1 y se repite el proceso, con lo que el algoritmo es muy rápido. Gráficamente:



Por ejemplo, una traza del reparto de cartas es el siguiente (están todas):

uno de espadas	sota de copas	rey de oros	seis de copas
seis de oros	caballo de bastos	cuatro de oros	sota de espadas
cuatro de espadas	caballo de oros	dos de espadas	rey de espadas
uno de oros	dos de oros	cuatro de bastos	cuatro de copas
tres de oros	uno de copas	dos de bastos	rey de copas
sota de bastos	caballo de espadas	siete de copas	siete de bastos
cinco de bastos	sota de oros	siete de espadas	tres de copas
rey de bastos	tres de bastos	uno de bastos	seis de espadas
dos de copas	caballo de copas	seis de bastos	cinco de espadas
cinco de copas	cinco de oros	tres de espadas	siete de oros

La generación de números aleatorios en C++ ha sido mejorada, de forma que ahora se separa el generador concreto de la distribución de probabilidad que se muestrea para obtener el valor aleatorio. Esto quiere decir que hay distribuciones uniformes, gaussianas, binomiales, etc. Se utilizará un código como este:

```
#include <random>
std::random_device rd;
std:: default_random_engine generador(rd());
std::uniform_int_distribution<> distribucion(0, 39);
auto val = distribucion (generador);
```

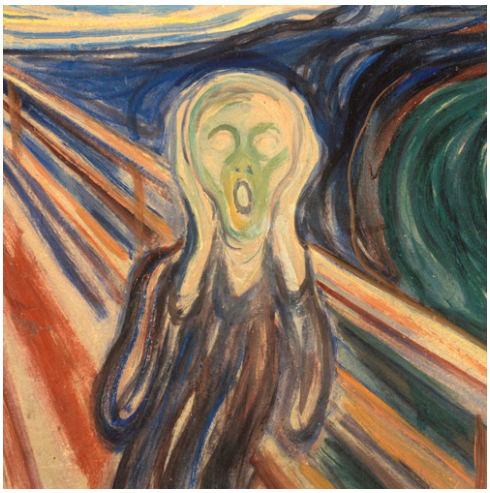
Si conoce las posiciones que quiere intercambiar del vector, la función `swap()` permite hacerlo fácilmente. Por ejemplo, para intercambiar los valores almacenados en los índices 0 y 20:

```
swap(a[0], a[20]);
```

Se pide que implemente los dos algoritmos y realice medidas de tiempos de ejecución. Utilice la clase **`Carta`** anterior para modelar las cartas de la baraja.

Las soluciones vienen a continuación.

Consultar solo si ya está un poco desesperad@, o para comparar soluciones



SOLUCIÓN EJERCICIO 1: MIRAR SOLO EN CASO DE DESESPERACIÓN

```
#include <iostream>
#include <cmath>

class Sphere {
   long diam;
public:
   Sphere (long el_diam) : diam = el_diam {}
   long get_diam () { return diam; }
   void set_diam (long el_diam) { diam = el_diam; }
   double volumen () { return M_PI * std::pow(diam,3) / 6.0; }
   double area () { return M_PI * diam * diam; }
   friend std::ostream& operator<< (std::ostream& os, Sphere& sp) {
       os << "Esfera de diametro: " << sp.diam;
       return os;
  }
};

int main () {
    Sphere sp1 = 1, sp2 {10};
    std::cout << "Esfera1: " << sp1 << " | Esfera2: " << sp2.get_diam() << std::endl;
    std::cout << "Area esfera1: " << sp1.area() << ", volumen esfera1: "
       << sp1.volumen() << std::endl;
    sp1.set_diam(100);
    std::cout << "Area esfera1: " << sp1.area() << ", volumen esfera1: "
       << sp1.volumen() << std::endl;
    long d;
    std::cin >> d;
    Sphere sp3 { d };
    std::cout << "Area esfera1: " << sp3.area() << ", volumen esfera1: "
       << sp3.volumen() << std::endl;
    return 0;
}
```

SOLUCIÓN EXTENSIÓN EJERCICIO 1: MIRAR SOLO EN CASO DE DESESPERACIÓN

```
#include <iostream>
#include <cmath>

namespace upct::tr::poo::punto { // ¿y por qué no?
    class Point_3D {
        long x_, y_, z_;
    public:
        Point_3D (long x=0, long y=0, long z=0) : x_{x}, y_{y}, z_{z} {}
        long get_x() const { return x_; }
        long get_y() const { return y_; }
        long get_z() const { return z_; }
        double distancia_punto (const Point_3D &p) const {
            double dx = p.x_ - x_;
            double dy = p.y_ - y_;
            double dz = p.z_ - z_;
            return sqrt (dx*dx + dy*dy + dz*dz);
        }
    };
    // ¿Podría declarar operator<< como función no friend? (tampoco es 100% necesario)
}
class Sphere {
   long diam;
   upct::tr::poo::punto::Point_3D centro_;
public:
   Sphere(long el_diam, long x=0, long y=0, long z=0) :
        diam_ {el_diam}, centro_{x, y, z} {}
   long get_diam () { return diam; }
   void set_diam (long el_diam) { diam = el_diam; }
   double volumen () { return M_PI * std::pow(diam,3) / 6.0; } // (*)
   double area () { return M_PI * diam * diam; } // (*)
   friend std::ostream& operator<< (std::ostream& os, Sphere& sp) { // (*)
       os << "Esfera de diametro: " << sp.diam;
       return os;
  }
  bool dentro_esfera (const upct::tr::poo::punto::Point_3D &p) { // (*)
      return ( p.distancia_punto(centro_) <= (diam_/2) );
  }
};
// ¿podría declarar las funciones marcadas con (*) fuera de la clase?
int main () {
    Sphere sp1 {1, 3, 4}; // creada en el punto (3, 4, 0)
    Sphere sp2 {10}; // creada en el origen de coordenadas
    std::cout << "Esfera1: " << sp1 << " | Esfera2: " << sp2.get_diam()
              << std::endl;
    std::cout << "Area esfera1: " << sp1.area() << ", volumen esfera1: "
              << sp1.volumen() << std::endl;
        upct::tr::poo::punto::Point_3D p_si {1,2,3}, p_no {12, 2, 3};
    std::cout << "\tp_si? " << sp2.dentro_esfera(p_si)
              << "\tp_no? " << sp2.dentro_esfera(p_no) << '\n';
    return 0;
}
```

SOLUCIÓN EJERCICIO 5: MIRAR SOLO EN CASO DE DESESPERACIÓN

```
namespace secreto {
   class Secret_Msg {
      std::string msg, clave;
      bool candado;
   public:
      Secret_Msg(const std::string &el_msg, const std::string &la_clave) :
         msg{ el_msg }, clave{ la_clave }, candado{ true } {}

      bool is_locked() { return candado; }

      bool unlock(const std::string &la_clave) {
         if (la_clave == clave) {
            candado = false;
            return true;
         }
         return false;
      }

      bool lock() {
         return lock(clave);
      }

      bool lock(const std::string &nueva_clave){
         if (!candado) {
            candado = true;
            clave = nueva_clave;
            return true;
         }
         return false;
      }

      bool erase() {
         if (!candado) {
            msg.clear();
            return true;
         }
         return false;
      }

      friend std::ostream& operator<< (std::ostream &os, const Secret_Msg &m) {
         if (!m.candado) {
            os << m.msg;
         }
         return os;
      }
   };

   int main_secret() {
      Secret_Msg m{"Mensaje secreto :-O", "123abc"};
      std::cout << '|' << m << "| cerrado? " << m.is_locked() << std::endl;
      std::cout << "unlock con clave mala? " << m.unlock("clave mala") << std::endl;
      std::cout << "unlock con clave buena? " << m.unlock("123abc") << std::endl;
      std::cout << '|' << m << '|' << std::endl;

      m.lock();
```

```
        std::cout << '|' << m << "| cerrado? " << m.is_locked() << std::endl;
        std::cout << "unlock con clave buena? " << m.unlock("123abc") << std::endl;
        std::cout << '|' << m << '|' << std::endl;

        m.lock("abc123");
        std::cout << '|' << m << "| cerrado? " << m.is_locked() << std::endl;
        std::cout << "unlock con clave antigua? " << m.unlock("123abc") << std::endl;
        std::cout << "unlock con clave nueva? " << m.unlock("abc123") << std::endl;
        std::cout << '|' << m << '|' << std::endl;
        m.erase();
        std::cout << '|' << m << '|' << std::endl;
        return 0;
    }
}
```

SOLUCIÓN EJERCICIO 6: MIRAR SOLO EN CASO DE DESESPERACIÓN

```
#include <iostream>

namespace cartas {
   enum class T_Palos {oros=0, copas, espadas, bastos};
   std::ostream& operator<< (std::ostream &os, const T_Palos &v) {
      static const char* nombres[] = { "OROS", "COPAS", "ESPADAS", "BASTOS" };
      os << nombres[static_cast<int>(v)];
      return os;
   }

   enum class T_Numeros {as, dos, tres, cuatro, cinco, seis, siete, sota, caballo, rey};
   std::ostream& operator<< (std::ostream &os, const T_Numeros &v) {
      static const char* nombres[] = { "AS", "DOS", "TRES", "CUATRO", "CINCO", "SEIS",
        "SIETE", "SOTA", "CABALLO", "REY" };
      os << nombres[static_cast<int>(v)];
      return os;
   }

   class Carta {
      T_Palos palo;
      T_Numeros numero;
   public:
      friend std::ostream& operator<< (std::ostream &os, const Carta &c) {
         os << c.numero << " de " << c.palo;
         return os;
      }

      Carta(T_Numeros n, T_Palos p) : numero{ n }, palo{ p } {}

      T_Palos get_palo() const { return palo; }

      T_Numeros get_numero() const { return numero; }
   };

   class Brisca {
      T_Palos muestra;
      const int orden[10] = {10, 1, 9, 2, 3, 4, 5, 6, 7, 8};
   public:
      Brisca(T_Palos p) : muestra{ p } {}

      int puntuacion_carta(const Carta &c) {
         switch (c.get_numero()) {
         case T_Numeros::dos:
         case T_Numeros::cuatro:
         case T_Numeros::cinco:
         case T_Numeros::seis:
         case T_Numeros::siete:
            return 0;
         case T_Numeros::sota:
            return 2;
         case T_Numeros::caballo:
            return 3;
         case T_Numeros::rey:
            return 4;
         case T_Numeros::tres:
            return 10;
```

```
            case T_Numeros::as:
               return 11;
            }
         }

         bool gana_mano(const Carta &mano, const Carta &otro) {
            if (mano.get_palo() == otro.get_palo()) {
               return orden[static_cast<int>(mano.get_numero())] >
                      orden[static_cast<int>(otro.get_numero())];
            }
            else if (otro.get_palo() == muestra) {
               return false;
            }
            return true;
         }
      };

      int main_brisca() {
         Brisca b{ T_Palos::espadas };
         Carta c1{ T_Numeros::cinco, T_Palos::bastos },
               c2{ T_Numeros::tres, T_Palos::bastos },
               c3{ T_Numeros::dos, T_Palos::espadas };
         std::cout << "Gana " << c1 << " contra " << c2 << "? " << b.gana_mano(c1, c2)
          << std::endl;
         std::cout << "Gana " << c2 << " contra " << c1 << "? " << b.gana_mano(c2, c1)
          << std::endl;
         std::cout << "Gana " << c1 << " contra " << c3 << "? " << b.gana_mano(c1, c3)
          << std::endl;
         std::cout << "Gana " << c3 << " contra " << c1 << "? " << b.gana_mano(c3, c1)
          << std::endl;
         return 0;
      }
   }
```

EJERCICIO 6

```
#include <random>
#include <iostream>
#include <array>
#include <string>
#include <vector>

class Carta {
    std::string numero_, palo_;
public:
    Carta (const std::string &numero, const std::string &palo) :
       numero_{numero}, palo_{palo} {}
    friend std::ostream& operator<< (std::ostream& os, const Carta &c) {
        os << c.numero_ << " de " << c.palo_;
        return os;
    }
};
```

```
int main () {
    const std::array<std::string,10> numeros {"uno", "dos", "tres", "cuatro",
       "cinco", "seis", "siete", "sota", "caballo", "rey"};
    const std::array<std::string,4> palos {"oros", "copas", "espadas", "bastos"};
    std::vector<Carta> baraja;
    baraja.reserve (45);
    for (int i=0; i<numeros.size(); ++i) {
        for (int j=0; j<palos.size(); ++j) {
            baraja.emplace_back(numeros[i], palos[j]);
        }
    }

    std::random_device rd;
    std:: default_random_engine gen(rd());
    int val;
    for (int i=39; i>=0; --i) {
        std::uniform_int_distribution<> dist(0, i);
        val = dist(gen);
        std::cout << baraja[val] << std::endl;
        std::swap (baraja[i], baraja[val]);
    }
    return 0;
}
```