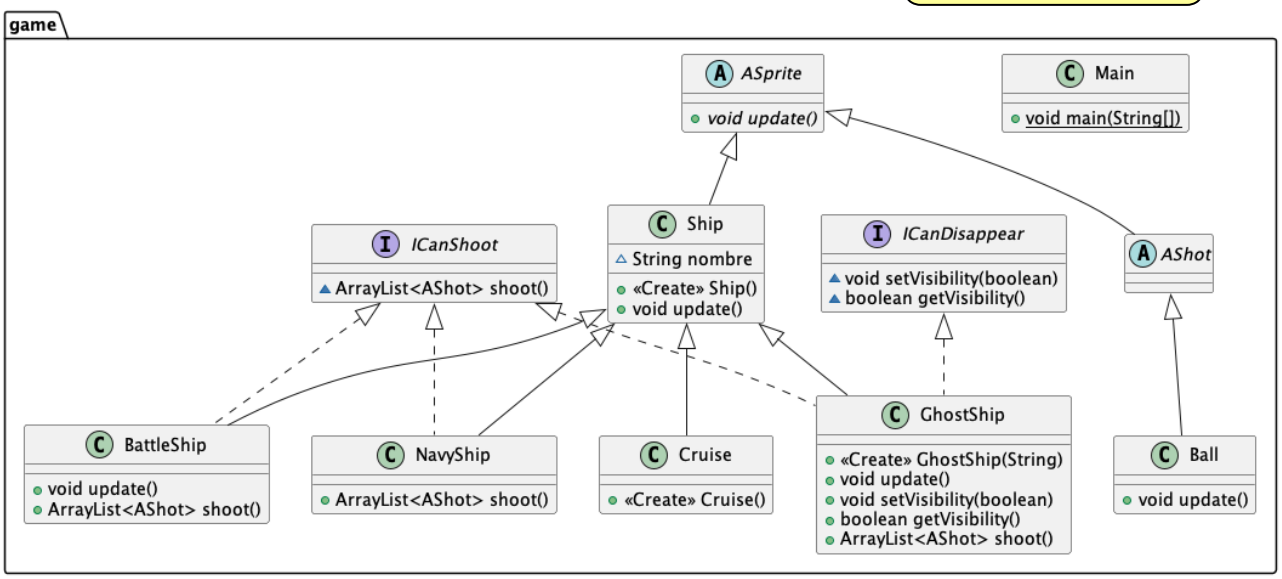
maría ana sanz CENTRO INTEGRADO	Ciclo: **C.F.G.S. Desarrollo de Aplicaciones Web** Módulo: **Programación 1º** Evaluación: **3ª** - **UT7 (Teoría)** - Fecha: **03/04/2023**
Nombre y Apellidos	**Nota**

Rodea con un círculo la respuesta correcta. Solo hay una en cada caso.
(*Correcto 1 pto./ Incorrecto -0,25 ptos.*).

Correctas 1 pto
Incorrectas -0,25 ptos
En blanco no puntúan



- Imagen 1 -

Respecto de la Imagen 1:

1. Selecciona la sentencia correcta:

   a) Ship es la clase abstracta de NavyShip.
   b) Cruise no implementa ninguna interfaz.
   c) Si el método update de ASprite es abstracto la clase AShot debe sobreescribirlo (override).
   d) GhostShip debe sobrescribir (override) los métodos update, shoot, setVisibility y getVisibility.

2. Selecciona la sentencia correcta:

   a) Si el método update de la clase ASprite es abstracto la clase ASprite tiene que ser abstracta.
   b) Si el método update de la clase ASprite es abstracto todas las clases hijas deben ser abstractas.
   c) Si el método update de la clase ASprite es abstracto ninguna clase hija puede ser abstracta.
   d) Si la clase ASprite es abstracta una clase hija no puede implementar una interfaz.

3. Si el método update de ASprite es abstracto, el método update de la clase Ship para llamarlo debe:

   a) no se puede hacer
   b) hacer un cast de tipo
   c) utilizar el operador super
   d) utilizar el operador this

4. Cual de las cuatro opciones no se puede realizar:

   a) Cruise cruise = new Ship();
   b) Ship navyShip = new NavyShip();
   c) ASprite ball = new Ball();
   d) ICanDisappear ghostShip = new GhostShip();

```
List<ICanShoot> list = new ArrayList<>();
list.add( new GhostShip("Yamato") );
list.add( new NavyShip() );
list.add( new BattleShip() );
list.add( (ICanShoot) new Cruise() );
list.add( new ICanShoot() {
    @Override
    public ArrayList<AShot> shoot() { return new ArrayList<>(); }
});
list.add( () -> new ArrayList<>() );
ArrayList<AShot> shots = new ArrayList<>();
for ( ICanShoot s : list ) { shots.addAll( s.shoot() ); }
for ( ICanShoot s : list ) { System.out.println( ((Ship)s).nombre ); }
```

- Código 1 -

Respecto del código 1 y la Imagen 1:

5. Hay un error en:

    a) list.add(new GhostShip());
    b) list.add(new NavyShip());
    c) list.add(new BattleShip());
    d) list.add((ICanShoot) new Cruise());

6. Hay un error en:

    a) list.add(() -> new ArrayList<>());
    b) ArrayList<AShot> shots = new ArrayList<>();
    c) for (ICanShoot s : list) { shots.addAll(s.shoot()); }
    d) for (ICanShoot s : list) { System.out.println(((Ship)s).nombre); }

```
List<Ship> list = new ArrayList<>();
list.add( new GhostShip("Yamato") );
list.add( new NavyShip() );
list.add( new BattleShip() );
list.add( new Cruise() );
list.add( new Ship() );
Collections.sort( list);
Collections.sort( list, new Comparator<Ship>() {
    @Override
    public int compare(Ship o1, Ship o2) {
        return -1 * o1.nombre.compareTo(o2.nombre);
    }
});
Collections.sort( list, (o1, o2) -> o1.nombre.compareTo(o2.nombre) );
Collections.reverse( list );
for ( Ship s : list ) { System.out.println( ((Ship)s).nombre ); }
```

- Código 2 -

Respecto del código 2 y la Imagen 1:

7. Hay un error en:

    a) Collections.sort(list);
    b) Collections.sort(list, new Comparator<Ship>() {
        @Override
        public int compare(Ship o1, Ship o2) {
            return -1 * o1.nombre.compareTo(o2.nombre);
        }
    });
    c) Collections.sort(list, (o1, o2) -> o1.nombre.compareTo(o2.nombre));
    d) Collections.reverse(list);

```
public class Ship extends ASprite {
    String nombre;
    public Ship() {
        nombre = getClass().getSimpleName();
    }
```

```
public GhostShip(String nombre) {
    super();
    this.nombre += "-" + nombre;
}
```

```
    List<Ship> list = new ArrayList<>();
    list.add( new GhostShip("Yamato") );
    list.add( new NavyShip() );
    list.add( new BattleShip() );
    list.add( new Cruise() );
    list.add( new Ship() );
    for ( Ship s : list ) {
        if (s instanceof ICanShoot) {
            System.out.println(
              ((Ship)s).nombre );
        }
    }
```

- Código 3 -

Respecto del código 3 y la Imagen 1:

8. Indica la secuencia de texto imprimida:

a) GhostShip-Yamato NavyShip BattleShip
b) GhostShip-Yamato NavyShip BattleShip Cruise Ship
c) GhostShip-Yamato NavyShip BattleShip Cruise
d) BattleShip Cruise NavyShip GhostShip-Yamato Ship

```
abstract class ASprite {
    int x;
    ASprite(){
        x = 50;
    }
    ASprite(int x){
        this.x = x;
    }
    abstract void update();
}
```

```
class Ship extends ASprite {
    int n = 5;
    Ship(){
        super(0);
    }
    Ship(int n){
        this.n = n;
    }
    @Override
    void update() {
        x += n;
    }
}
```

```
class Cruise extends Ship {
    Cruise(){
        n = 2*n;
    }
    Cruise(int n) {
        super(2*n);
    }
}
```

```
        ArrayList<ASprite> list = new ArrayList<>();
        list.add( new Ship() );
        list.add( new Cruise() );
        list.add( new Ship(10) );
        list.add( new Cruise(10) );
        for (ASprite sprite : list) { sprite.update(); }
        for (ASprite sprite : list) { System.out.printf(sprite.x + " "); }
```

- Código 4 -

Respecto del código 4 (sin tener en cuenta la Imagen 1):

9. El texto imprimido será:

a) 5 10 60 70
b) 55 60 20 40
c) 0 0 20 40
d) 50 50 20 40

```
public abstract class ASprite implements
Comparable<ASprite> {

    int x;

    abstract void update();

    @Override
    public int compareTo(ASprite other){
        return this.x - other.x;
    }

    @Override
    public String toString(){
        String s = super.toString();
        int i = s.lastIndexOf(".");
        s = s.substring(i+1);
        int j = s.indexOf("@");
        return s.substring(0, j+1) + x;
    }

}
```

```
public class Ship extends ASprite {

    int n;

    Ship(int n, int x){
        this.n = n;
        this.x = x;
    }

    @Override
    void update() {
        x += n;
    }
}
```

```
public class Cruise extends Ship {
    Cruise(int n, int x) {
        super(2*n, x);
    }
}
```

```
        ArrayList<ASprite> list = new ArrayList<>();
        list.add( new Ship(3, 2) );
        list.add( new Cruise(2, 0) );
        list.add( new Cruise(2, 4) );
        list.add( new Ship(2, 4) );
        for (ASprite sprite : list) { sprite.update(); }
        Collections.sort(list);
        for (ASprite sprite : list) { System.out.println(sprite.toString()); }
```

- Código 5 -

Respecto del código 5 (sin tener en cuenta la Imagen 1):

10. El texto imprimido será:

    a) Ship@5 Cruise@4 Cruise@8 Ship@6
    b) Cruise@4 Ship@5 Ship@6 Cruise@8
    c) Cruise@4 Cruise@8 Ship@5 Ship@6
    d) Cruise@8 Cruise@4 Ship@6 Ship@5

```
public class Ship {
    int n;
    int x;
    Ship(int n){
        this.n = n;
    }
    void update() {
        x += n;
    }
    @Override
    public boolean equals(Object o) {
        return o != null &&
               x == ((Ship)o).x;
    }
    @Override
    public int hashCode() {
        return x;
    }
}
```

```
    HashSet<Ship> ships = new HashSet<>();

    ships.add(new Ship(2));
    ships.add(new Ship(4));
    ships.add(new Ship(3));

    Ship ship = new Ship(2);
    ship.update();
    ships.add(ship);

    System.out.println( ships.size() );
```

- Código 6 -

Respecto del código 6 (sin tener en cuenta la Imagen 1):

11. El texto imprimido será:

a) 4 b) 3 c) 2 d) 1

```
public class Ship implements
Comparable<Ship> {
    int x;
    Ship(int x){
        this.x = x;
    }
    void update(int n) {
        x += n;
    }
    @Override
    public int compareTo(Ship other){
        return this.x - other.x;
    }
}
```

```
    TreeSet<Ship> ships = new TreeSet<>();

    ships.add(new Ship(4));
    ships.add(new Ship(2));

    Ship ship = new Ship(2);
    ship.update(2);
    ships.add(ship);

    ships.add(new Ship(3));

    for (Ship s : ships) {
        System.out.print(s.x + " ");
    }
```

- Código 7 -

Respecto del código 7 (sin tener en cuenta la Imagen 1):

12. El texto imprimido será:

a) 4 2 4 3
b) 4 2 3
c) 2 4 3
d) 2 3 4

```
public interface ICanFight {
    default void fight() {
        System.out.println("Zum!");
    }
}
```

```
public class Cruise implements ICanFight {
    @Override
    public void fight(){
        System.out.println("Bam!");
    }
}
```

```
public class BattleShip implements
ICanFight{
    @Override
    public void fight(){
        System.out.println("Pim!");
    }
}
```

```
public class NavyShip implements ICanFight{
}
```

```
public class GhostShip extends BattleShip{
}
```

```
    ArrayList<ICanFight> list = new ArrayList<>();
    list.add(new Cruise());
    list.add(new BattleShip());
    list.add(new NavyShip());
    list.add(new GhostShip());
    for (ICanFight ship : list) {
        ship.fight();
    }
```

- Código 8 -

Respecto del código 8 (sin tener en cuenta la Imagen 1):

13. El texto imprimido será:

a) Bam! Pim! Zum! Pim!
b) Bam! Pim!
c) Bam! Pim! Pim!
d) Zum! Zum! Zum! Zum!

```
public class Ship {
    int x;
    public void update() {
        x ++;
    }
}
```

```
public class BattleShip extends Ship{
    @Override
    public void update() {
        super.update();
        super.update();
    }
}
```

```
public class GhostShip extends BattleShip{
    @Override
    public void update() {
        super.update();
        super.update();
    }
}
```

```
Ship s1 = new Ship();
Ship s2 = new BattleShip();
Ship s3 = new GhostShip();

s1.update();
s2.update();
s3.update();

System.out.println(s1.x);
System.out.println(s2.x);
System.out.println(s3.x);

s3.update();

System.out.println(s3.x);
```

- Código 9 -

Respecto del código 9 (sin tener en cuenta la Imagen 1):

14. El texto imprimido será:

a) 1 2 3 4
b) 1 2 4 8
c) 1 2 2 4
d) 1 3 5 7

```
public class Ship {
    int x;
    public void update() {
        x ++;
    }
}
```

```
public class NavyShip extends Ship {
    @Override
    public void update() {
        x += 2;
    }
}
```

```
public class GhostShip extends Ship{
    @Override
    public void update() {
        x += 4;
    }
}
```

```
Ship s1 = new Ship();
s1.update();

NavyShip s2 = new NavyShip();
s2.update();

s1 = s2;
s1.update();

GhostShip s4 = new GhostShip();

Ship s3 = s4;
s3.update();

s4 = (GhostShip) s3;
s4.update();

System.out.println(s1.x + s2.x + s3.x + s4.x);
```

- Código 10 -

Respecto del código 10 (sin tener en cuenta la Imagen 1):

15. El texto imprimido será:

a) 24 b) 16 c) 14 d) 9