- **What's new in Java 8? Explain some of them.**
  Interface default and static methods. Antes de java 8, las interfaces no podían tener métodos estáticos o por defecto. A partir de esta nueva versión, podemos implementar este tipo de métodos en las interfaces.

  La forma de referenciar a un método, estático o no. A partir de java 8, se añade una nueva forma de referenciar a un método, utilizando "::", tanto para un método estático o uno no estático (User::getAge) o si no fuera estático (userA::getAge).

  Optional class. Antes de java 8 había que tener cuidado con los null pointer exception, y el chequeo de estos podría conllevar código repetido. Java 8 nos ofrece esta clase que hace de contenedor para evitar estos null pointer.

  Lambdas. Es una función anónima, un método que solo esta definido en una interfaz, pero no implementado. Al no estar implementado el programador puede implemntarla donde quiera.

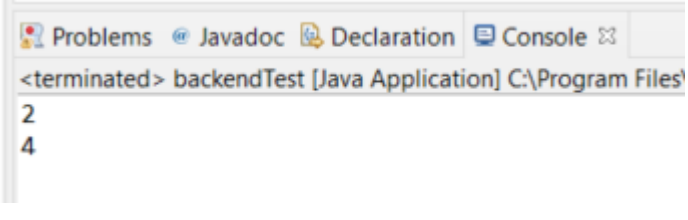
  Stream. Una nueva API que nos permite realizar operaciones sobre colecciones usando el modelo filtro/mapeo/reducción. Esto significa que se seleccionan los datos, se convierten al tipo deseado, y posteriormente se obtiene el resultado.

  Java time. Nueva API para tratar fechas, tiempos, instantes y duraciones. LocalDate, LocalDateTime, LocalTime.

  Algunos métodos nuevos en clases como String o Integer. Método join, para unir dos cadenas, métodos para dividir y comparar números, y las operaciones de suma, resta, máximo y mínimo.

- **Given the following list implement a solution in order to get even numbers using Java 8 Streams**
  List<Integer> list = Arrays.asList(1,2,3,4);

```
public static void main(String[] args) {
    List<Integer> list = Arrays.asList(1, 2, 3, 4);
    List<Integer> even = list.stream().filter(i -> i % 2 == 0).collect(Collectors.toList());

    even.forEach(System.out::println);
}
```



- **What do you notice when you do code review?**
  Normalmente me doy cuenta de cosas que podrían estar mejor hechas, siguiendo buenas prácticas de programación, más moduladas y/o explicadas.

- **Have you ever worked with Scrum? Tell us what it is, what events do you remember and what roles are involved?**
  He trabajado con scrum aunque no de forma estricta. Es una metodología ágil que intenta mediante una forma de trabajar, obtener el mejor resultado de un proyecto. Se realizan entregas parciales y regulares del producto, normalmente unas dos semanas. Para cada sprint se recogen unos requisitos a llevar a cabo, se planifican y se asignan a cada uno de los componentes del equipo. Al final de cada sprint se hace una reunión para valorar la manera de trabajar y los problemas encontrados.
- **What access modifiers (or visibility) do you know in Java?**
  Public, private, protected y default.
  Public es visible por todas las clases y paquetes.
  Protected es visible por otras clases dentro del mismo paquete y por subclases que estén en otro paquete.
  Private es solo visible dentro de su clase.
  Default es visible dentro de su clase y su paquete.

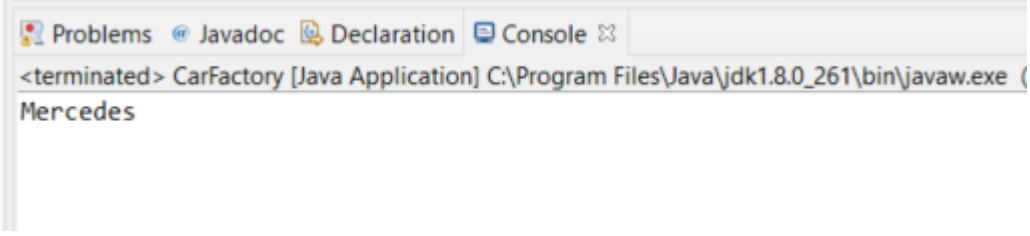
- **Differences between an abstract class and an interface. When would you use one or the other?**
  Una clase solo puede extender de una sola clase abstracta, sin embargo, puede implementar varias interfaces (herencia múltiple).
  Una clase abstracta puede tener variables de instancia, sin embargo, esto no es posible en una interfaz.
  Una clase abstracta puede ser publica, privada, protected pero una interfaz solo es pública.
  Una clase abstracta puede contener constructores al contrario que las interfaces.

- **What is Maven and why is it used? What is Maven life cycle?**
  Es una herramienta que simplifica el proceso de compilar y generar los ejecutables a partir del código fuente. Además, puede encargarse de la gestión de dependencias, despliegue en el servidor y ejecución de pruebas y generación de informes y/o documentación.
- **What is Git and what is it used for? List all Git commands that you know.**
  Es un software para el control de versiones. Se registran los cambios en los fuentes para poder coordinar entre los distintos miembros del equipo el desarrollo.
  Git init
  Git merge <rama>
  Git pull
  Git commit
  Git push
  Git status
  Git add
- **What is a mock? What would you use it for?**
  Se trata de aislar un objeto que tiene dependencias con otro para poder realizar un test. Para aislar este objecto “mockeamos” esas dependencias para simular el comportamiento real.
- **How would you explain to someone what Spring is? What can it bring to their projects?**
  Es un framework de código abierto basado en java para crear aplicaciones empresariales.

Spring trabaja con ORM que nos facilitan las operaciones con la base de datos, preocupándonos solo de las querys más complejas.
Es un framework basado en el modelo vista controlador.
Proporciona gestión de transacciones.
Permite separar el registro de log, la auditoria, la seguridad y el almacenamiento en caché de la lógica de negocio a través de la programación orientada a aspectos.

- **What's the difference between Spring and Spring Boot?**
  Springboot es una extensión de spring. Esta nos permite centrarnos en la aplicación a crear dejando las tareas de configuración a un lado (selección de jars, despliegue en servidor...).
- **Do you know what CQRS is? And Event Sourcing?**
- **Differences between IaaS and PaaS. Do you know any of each type?**
- **Explain what a Service Mesh is? Do you have an example?**
- **Explain what is TDD? What is triangulation?**
- **Apply the Factory pattern with lambda expressions**

```
class CarFactory {
    static final Map<String, Supplier<Car>> map = new HashMap<>();
    static {
        map.put("Mercedes", Mercedes::new);
        map.put("Mazda", Mazda::new);
    }

    public static Car create(String carType) {

        Supplier<Car> shape = map.get(carType);
        if (shape != null) {
            return shape.get();
        }
        throw new IllegalArgumentException("No existe esta marca de coche " + carType);
    }

    public static void main(String args[]) {
        Car car = CarFactory.create("Mercedes");
        System.out.println(car);
    }

    public static class Mazda implements Car {
        public String toString() {
            return "Mazda";
        }
    }

    public static class Mercedes implements Car {
        public String toString() {
            return "Mercedes";
        }
    }
```



- **Reduce the 3 classes (OldWayPaymentStrategy, CashPaymentStrategy and CreditCardStrategy) into a single class (PaymentStrategy). You do not need to create any more classes or interfaces. Also, tell me how you would use PaymentStrategy, i.e. the different payment strategies in the Main class**

```
public interface PaymentStrategy {

    public static double payCash(double amount) {
        double serviceCharge = 5.00;
        return amount + serviceCharge;
    }

    public static double payCreditCard(double amount) {
        double serviceCharge = 5.00;
        double creditCardFee = 10.00;
        return amount + serviceCharge + creditCardFee;
    }
}
```

```
        System.out.println(PaymentStrategy.payCash(50));
        System.out.println(PaymentStrategy.payCreditCard(30));
```