En el siguiente código

```
class A {}
class B extends A {}

...
A obj = new B();
```

Seleccione una:

- a. El tipo en tiempo de compilación de 'obj' es A y su tipo en tiempo de ejecución es B. ✔
- b. El tipo en tiempo de compilación de 'obj' es A y su tipo en tiempo de ejecución también es A.
- c. El tipo en tiempo de compilación de 'obj' es B y su tipo en tiempo de ejecución es A.
- d. El tipo en tiempo de compilación de 'obj' es B y su tipo en tiempo de ejecución también es B.

Indica en que fase del proceso de compilación se realiza la comprobación de tipos.

Seleccione una:

- a. Análisis sintáctico
- b. Análisis semántico ✔
- c. Generacíon de código
- d. Análisis léxico

Los atributos estáticos

Seleccione una:

- a. se almacenan en la memoria asignada a cada instancia de la clase donde se definen
- b. se almacenan en la pila
- c. se almacenan en la memoria asignada al objeto Class de la clase donde se definen ✔

Indica la opción correcta para la máquina virtual de Java (JVM):

Seleccione una:

- a. En la pila se almacenan cualquier variable local u objeto si el método que se está ejecutando es un método estático.
- b. En el heap se almacenan cualquier variable local u objeto si el método que se está ejecutando es un método de instancia.
- c. Las variables locales (tipos primitivos, referencias a objetos) se almacenan en la pila y los objetos en el heap ✔
- d. Las variables locales (tipos primitivos, referencias a objetos) se almacenan en el heap y los objetos en la pila

Dado este código:

```
public class Animal {
	public void come() {
		System.out.println("Animal, ¡come!");
	}
}
```

```
public class Serpiente extends Animal {
	public void come() {
		System.out.println("Serpiente, ¡come!");
	}
	public void repta() {
		System.out.println("¡Repta!");
	}
}
```

```
public class Ave extends Animal {
	public void mudaPlumas() {
		System.out.println("Ave, ¡muda las plumas!");
	}
	public void come() {
		System.out.println("Ave, ¡come!");
	}
}
```

```
public class AveVoladora extends Ave {
	public void vuela() {
		System.out.println("¡Vuela!");
	}
}
```

```
public class Pinguino extends Ave {
	public void come() {
		System.out.println("Pingüino, ¡come!");
	}
}
```

```
public class Cliente {
	public Animal F(char c) {
		Animal animal = null;

		switch (c) {
			case 'a':
				animal = new Ave();
				break;
			case 'v':
				animal = new AveVoladora();
				break;
			case 'p':
				animal = new Pinguino();
				break;
			case 's':
				animal = new Serpiente();
				break;
			default:
				animal = new Animal();
		}

		return animal;
	}
}
```

Indica la opción correcta

```
public class Main {
	public static void main(String[] args) {
		Cliente m = new Cliente();
		Animal a = null;
		if (Math.random()>0.5)
		     a = m.F('s');
		else
		    a = m.F('p');

		Serpiente s = (Serpiente) a;
		s.repta();
	}
}
```

Podría fallar en tiempo de ejecución

```
public class Main {
	public static void main(String[] args) {
		Animal a = new Serpiente();
		a.come();
	}
}
```

Se está haciendo un upcasting ✔

```
public class Main {
	public static void main(String[] args) {
		Cliente m = new Cliente();
		Animal a = m.F('s');
		a.repta();
	}
}
```

No compila ✔

```
public class Main {
	public static void main(String[] args) {
		Cliente m = new Cliente();
		Animal a = m.F('s');
		if (a instanceof Serpiente) {
			Serpiente s = (Serpiente) a;
			s.repta();
		}
	}
}
```

Se está haciendo un downcasting seguro ✔

Dado este código:

```
public class Animal {
	public void come() {
		System.out.println("Animal, ¡come!");
	}
}
```

```
public class Serpiente extends Animal {
	public void come() {
		System.out.println("Serpiente, ¡come!");
	}
	public void repta() {
		System.out.println("¡Repta!");
	}
}
```

```
public class Ave extends Animal {
	public void mudaPlumas() {
		System.out.println("Ave, ¡muda las plumas!");
	}
	public void come() {
		System.out.println("Ave, ¡come!");
	}
}
```

```
public class AveVoladora extends Ave {
	public void vuela() {
		System.out.println("¡Vuela!");
	}
}
```

```
public class Pinguino extends Ave {
	public void come() {
		System.out.println("Pingüino, ¡come!");
	}
}
```

```
public class Main {
        public void F(char c) {
                Animal animal = null;

                switch (c) {
                        case 'a':
                                animal = new Ave();
                                break;
                        case 'v':
                                animal = new AveVoladora();
                                break;
                        case 's':
                                animal = new Serpiente();
                                break;
                        case 'p':
                                animal = new Pinguino();
                                break;
                        default:
                                animal = new Animal();
                }

                animal.come();
        }
}
```

Indica lo que se imprime por pantalla al llamar al método F(char c) dadas las siguientes opciones

c='s'	Serpiente, ¡come!	✔
c = 'v'	Ave, ¡come!	✔
c = 'p'	Pingüino, ¡come!	✔
c = 'a'	Ave, ¡come!	✔

Indica si las siguientes afirmaciones son correctas.

El siguiente código es correcto en una **comprobación de tipos dinámica**

```
class A {}
class B extends A { int b; }
a = new B();
a.b;
```

Cierto ✔

Sabemos que este código de Python da error. Este lenguaje usa un sistema de comprobación de tipos dinámico. Por tanto, el error lo da porque Python usa un tipado fuerte.

```
x = "Hola"
y = 2019
print x + y
```

Cierto ✔

Todo lenguaje que use una comprobación de tipos dinámica usa tipado débil

Falso ✔

Sabemos que este código de Javascript no da errores sintácticos. Este lenguaje usa un sistema de comprobación de tipos dinámico. Por tanto, puede funcionar porque tiene un sistema de tipos débil.

```
js> x="5" <-- String
5
js> x=x*5 <-- Int
25
```

Cierto ✔

Todo lenguaje que use una comprobación de tipos estática usa tipado fuerte

Falso ✔

El siguiente código es correcto en una **comprobación de tipos estática**

```
class A {}
class B extends A { int b; }
A a = new B();
a.b;
```

Falso ✔

Siguiendo el orden de los números en los comentarios, sigue la traza de ejecución del código desde el programa principal y escribe las palabras correctas en los huecos.

Las palabras válidas son:

```
stack
heap
compilación
ejecución
```

NOTA: se han omitido algunos artículos en las oraciones para evitar dar pistas adicionales a las respuestas.

```
public class Personaje {
        protected String nombre;
        public Personaje(String nombre) {this.nombre = nombre;}
        // 19º. Se añade a  stack  ✔  el constructor de Personaje

        // 20º. Se reserva en  stack  ✔  memoria para el parámetro 'nombre'

        // 21º. Se asigna en  heap  ✔  la referencia 'this.nombre' al valor del parámetro 'nombre'

        // 22º Cuando acaba este constructor de Personaje se elimina de  stack  ✔


        public String getNombre() {return this.nombre;}
        public String expresate() {return "";}
}
```

```
public class Animal extends Personaje {
	Extremidad[] extremidades; // composicion
	protected Animal(String nombre, Extremidad [] extremidades) {
	// 17º. Se añade a stack ✔ el constructor de Animal

	// 18º. Se reserva en stack ✔ memoria para los parámetros 'nombre' y 'extremidades'
		super(nombre);

		this.extremidades = new Extremidad[extremidades.length];
		// 23º. Se reserva en heap ✔ la memoria para un array de Extremidades

		// 24º. Se asigna en heap ✔ la referencia 'this.extremidades' al objeto anterior
		for (int i=0; i < extremidades.length; i++) {
			this.extremidades[i] = extremidades[i].clona();
			// En tiempo de compilación ✔ se comprueba que la clase Extremidad
			// tiene un método clona() que retorna un tipo Extremidad o compatible
			// Cuando i=0, como el tipo de extremidades[i] en tiempo de ejecución ✔
			// es Brazo, se invoca al método clona() de Brazo

			//Para i=1 se repiten los pasos 25º, 26º y 27º
			//Para i=2 y luego para i=3 se repiten los pasos 28º, 29º y 30º
		}
	}
	// 31º. Se elimina de stack ✔ el constructor de Animal

}
```

```
        public String expresate() {
                String frase = frases.get((int) (Math.random() * frases.size()));
                StringBuilder sb = new StringBuilder();
                sb.append(frase);
                for (Animal mascota: mascotas) {
                        sb.append(',');
                        sb.append(mascota.expresate());
                }
                return sb.toString();
        }

        public void addMascota(Animal animal) {
                mascotas.add(animal);
        }
}
```

```
public class Perro extends Animal {
        protected Perro(String nombre) {
                super(nombre, new Extremidad[] {new Pata(), new Pata(), new Pata(), new Pata()});
        }
        public String expresate() {return "Guau";}
}
```

```
public class Extremidad {
        public Extremidad clona() {return new Extremidad();}
}
```

```
public class Brazo extends Extremidad {
        public Extremidad clona() {return new Brazo();}
        // 25º. Se añade a stack ✔ el método clona()
        // 26º. Se reserva en heap ✔ la memoria para la nueva instancia de Brazo
        // El tipo en tiempo de ejecución ✔ del valor devuelto por el método clona() es Brazo
        // El tipo en tiempo de compilación ✔ del valor devuelto por el método clona() es Extremidad
        // Se invoca al constructor por defecto de Brazo, luego al de Extremidad
        // 27º. Se elimina de stack ✔ el método clona()
}
```

```
public class Pierna extends Extremidad {
        public Extremidad clona() {return new Pierna();}
        // 28º. Se añade a  stack  ✔  el método clona()
        // 29º. Se reserva en  heap  ✔  la memoria para la nueva instancia de Pierna
        // El tipo en tiempo de  ejecución  ✔  del valor devuelto por el método clona() es Pierna
        // El tipo en tiempo de  compilación  ✔  del valor devuelto por el método clona() es Extremidad
        // Se invoca al constructor por defecto de Pierna, luego al de Extremidad
        // 30º. Se elimina de  stack  ✔  el método clona()
}
```

```
public class Pata extends Extremidad {
        public Extremidad clona() {return new Pata();}
}
```

```
public class Cuento {
        protected List personajes;

        public Cuento() {
                // 3º. Se añade a stack ✔ el constructor
                creaPersonajes();
        }

        public void creaPersonajes() {
        // 4º. Se añade a stack ✔ el método creaPersonajes()
                personajes = new ArrayList<>();
                // 5º. Se reserva en heap ✔ la memoria para un objeto de tipo ArrayList
                // 6º. Se enlaza en el heap ✔ la referencia 'personajes' de esta instancia

                List<String> frasesLucky = new ArrayList<>();
                // 7º. Se reserva en heap ✔ la memoria para un objeto de tipo ArrayList<String>
                // 8º. Se reserva en stack ✔ la memoria para la referencia 'frasesLucky'
    frasesLucky.add("Soy un pobre vaquero solitario");
    frasesLucky.add("Estoy lejos de mi hogar");

                Humano lucky = new Humano("Lucky", frasesLucky);
                // 9º. Se reserva en heap ✔ la memoria de esta instancia de tipo Humano

                personajes.add(lucky);
                // 39º. Se añade a stack ✔ el método add de ArrayList,
                // donde el tipo de su argumento proporcionado en tiempo de compilación ✔ es Personaje
                // y en tiempo de ejecución ✔ es Humano

                // El ejercicio acaba en este paso 39º - se invita al alumno a continuar con el resto de traza

                Perro rantamplan = new Perro("Rantamplán");
                lucky.addMascota(rantamplan);
                personajes.add(rantamplan);
                personajes.add(new Arbol("Árbol del desierto"));
        }
```

```
public void personajesSeExpresan() {
        for (Personaje personaje: personajes) {
                System.out.println(personaje.getNombre());
                System.out.println("\t" + personaje.expresate());
        }
}

public static final void main(String [] args) {
        // 1º. Se añade a  stack  ✔  el método estático main
        Cuento cuento = new Cuento();
        // 2º. Se reserva en  heap  ✔  la memoria para un nuevo objeto de tipo Cuento
        cuento.creaPersonajes();
        cuento.personajesSeExpresan();
}
```

Dado este código:

```
public class Animal {
        public void come() {
                System.out.println("Animal, ¡come!");
        }
}
```

```
public class Serpiente extends Animal {
        public void come() {
                System.out.println("Serpiente, ¡come!");
        }
        public void repta() {
                System.out.println("¡Repta!");
        }
}
```

```
public class Ave extends Animal {
        public void mudaPlumas() {
                System.out.println("Ave, ¡muda las plumas!");
        }
        public void come() {
                System.out.println("Ave, ¡come!");
        }
}
```

```
public class AveVoladora extends Ave {
        public void vuela() {
                System.out.println("¡Vuela!");
        }
}
```

```
public class Pinguino extends Ave {
        public void come() {
                System.out.println("Pingüino, ¡come!");
        }
}
```