0) O que o código abaixo faz?

```
boolean doidao(char c) {
	boolean resp = false;
	int v = (int)c;

	if (v == 65 || v == 69 || v == 73 || v == 79 || v == 85 || v == 97 ||
		v == 101 || v == 105 || v == 111 || v == 117) {
		resp = true;
	}

	return resp;
}
```

R: O código verifica se o caractere recebido como parâmetro é uma vogal ou não. Para isso ele utiliza os valores representacionais para vogal minúsculo ou maiúsculo da tabela ASCII.

1) Qual das duas versões é mais fácil de entender?

A -

```
boolean isConsoante(String s, int i) {
	boolean resp = true;
	if (i == s.length()) {
		resp = true;
	}else if (isConsoante(s.charAt(i)) == false) {
		resp = false;
	}else {
		resp = isConsoante(s, i + 1);
	}
	return resp;
}
```

B -

```
boolean isConsoante(String s, int i) {
	boolean resp = true;
	if (i < s.length()) {
		if (!isConsoante(s.charAt(i))) {
			resp = false;
		}else {
			resp = isConsoante(s, i + 1);
		}
	}else {
		resp = true;
	}
	return resp;
}
```

R: A implementação ‘A’, pois está mais legível sobre a possibilidade de a recursão está verificando se a string fornecida possui pelo menos uma consoante.

2) Qual é a sua opinião sobre o código REAL abaixo?

```
Unidade recuperarUnidadeComCodigoDeUCI(Unidade unidadeFilha) {
	Unidade retorno = null;

	if (unidadeFilha.getCodUci() != null && !unidadeFilha.getCodUci().isEmpty()) {
		retorno = unidadeFilha;
	}else {
		retorno = unidadeFilha.getUnidadeSuperior();
	}

	while (retorno == null || retorno.getCodUci() == null || retorno.getCodUci().isEmpty()) {
		retorno = retorno.getUnidadeSuperior();
	}

	return retorno;
}
```

R: O código fornecido possui difícil compreensão devido a baixa clareza dos métodos do código e a falta de comentários que facilitariam um pouco a legibilidade do código. Provavelmente a leitura desse código pode ser difícil até mesmo para quem conhece o contexto do software onde ele está implementado.

3) Qual é a diferença entre os dois métodos abaixo?

```
int m1(int i) {
	return i--;
}

int m2(int i) {
	return --i;
}
```

R: O primeiro trecho de código realiza um pós incremento da variável ‘i’, retornando o mesmo valor recebido por parâmetro, enquanto o segundo realiza o pré incremento, retornando para o método que o chamou o valor recebido acrescido de uma unidade.

4) O que o programa abaixo mostra na tela?

```
byte b = 0; short s = 0; int i = 0; long l = 0;
while (true) {
	b++; s++; i++; l++;
	System.out.println(b + “ ” + s + “ ” + i + “ ” + l);
}
```

R: O código exibe “infinitamente” o valor presente nas quatros variáveis ‘byte b’, ‘shot s’, ‘int i’ e ‘long l’, em algum momento ocorrerá overflow em todas as variáveis, que é quando o limite máximo de representação de determinado tipo é atingido, reiniciando a contagem. O código é interrompido com a interferência do usuário ou erro de execução.

5) Por que o código abaixo imprime [46 - 11]?

```
int x = 23, y = 23;
x = x << 1;
y = y >> 1;
System.out.println(“[” + x + “ - ” + y + “]”);
```

R: Os operadores de shift esquerda (<<) e shift direita (>>) realiza o deslocamento binário nessas direções, sendo o 23 equivalente a 0001_0111 em binário temos o seguinte resultado ao realizar as operações: x << 1 : 0010_1110 (46) e y >> 1 : 0000_1011 (11). Por esse motivo é impresso [46 - 11].