67. Como as mudanças de design e paradigma em uma linguagem de programação impactaram a forma como escrevemos programas? Escolha uma linguagem de sua preferência, identifique duas versões ou eras distintas (antiga vs. moderna) e demonstre, através de trechos de código comparativos, como uma mesma tarefa era resolvida no passado e como é resolvida hoje, explicando as vantagens e desvantagens de cada abordagem.

O Java 2 (JDK 1.2), lançado em 1998, institucionalizou o *Java Collections Framework* (JCF), substituindo as estruturas *ad hoc* das versões anteriores, que se baseavam em classes como Vector, Hashtable e na interface Enumeration. Durante a fase inicial do JCF (JDK 1.2 a 1.4), os mecanismos padrão para percorrer coleções restringiam-se à implementação explícita da interface Iterator ou à iteração via índice utilizando laços for convencionais.

```
List<String> nomes = new ArrayList<>();
nomes.add("Ana");
nomes.add("Bruno");
nomes.add("Carlos");
// Abordagem com Iterator (Introduzida no Java 2)
Iterator<String> iterator = nomes.iterator();
while (iterator.hasNext()) {
	String nome = iterator.next();
	System.out.println(nome);
}
// Ou a abordagem antiga por índice (menos segura para genéricos na época)
for (int i = 0; i < nomes.size(); i++) {
	String nome = (String) nomes.get(i);
	System.out.println(nome);
}
```

Essa abordagem era verbosa, exigindo múltiplas linhas de código apenas para realizar uma iteração simples, além de obrigar o desenvolvedor a gerenciar manualmente o estado do cursor. Com a evolução da linguagem, o Java trouxe duas revoluções fundamentais na forma de percorrer coleções: o *Enhanced For-Loop* e as *Streams* com *Lambdas*.

Java 5: Enhanced For-Loop
Conhecido também como *for-each*, este recurso representou um avanço crucial na abstração de baixo nível. Ao ocultar a complexidade do Iterator, ele deslocou o foco da lógica de controle para a lógica de negócio. Combinada com a introdução dos *Generics*, o *for-each* erradicou a propensão a erros de IndexOutOfBoundsException e removeu a necessidade de conversões de tipo (*casts*) explícitas, elevando a robustez e a clareza do código.

```
// Sintaxe limpa e legível
for (String nome : nomes) {
	System.out.println(nome);
}
```

Streams e Lambdas (Java 8+)
Representando a evolução mais profunda no processamento de dados em Java, esta introdução marcou a transição do estilo imperativo para o declarativo. Em vez de instruir explicitamente como iterar, o desenvolvedor define o que deve ser processado. Isso não apenas alinha o código mais estreitamente à lógica de negócios, mas também habilita a composição de pipelines de dados (filtragem, mapeamento, redução) de forma concisa, erradicando a verbosidade de variáveis intermediárias e a complexidade de iterações aninhadas.

```
// Estilo Funcional com Stream
nomes.stream()
	.filter(nome -> !nome.isEmpty()) // Filtragem intermediária
	.map(String::toUpperCase) // Transformação
	.forEach(System.out::println); // Ação final
```