Explicación del Código:Simple Repetition

En este documento vamos a analizar la implementación en Java del problema número 7, Simple Repetition, y veremos dos enfoques, Método Cándido (lento) y método Óptimo (rápido y enviado a la plataforma)

1. Método Cándido

Este método intenta resolver el problema tal cual está descrito:

toma el número X, lo repite K veces, crea un número gigantesco y luego verifica si ese número es primo.

Generar el número concatenado, usamos un StringBuilder:

```
StringBuilder numeroStr = new StringBuilder();
for (int i = 0; i < k; i++) {
   numeroStr.append(x);
}
```

Luego lo convertimos a un número long: long numUnido = Long.parseLong(numeroStr.toString());

Verificar si ese número es primo

Recorremos desde 2 hasta numUnido - 1 verificando residuos:

```
boolean primo = true;
for (long i = 2; i < numUnido; i++) {
   if (numUnido % i == 0) {
      primo = false;
      break;
   }
}
return primo;
```

Problemas del método Cándido

- Cuando X es grande, repetirlo K veces genera números imposibles de guardar.
- Incluso si cabe en un long, comprobar primalidad desde 2 hasta numUnido es terriblemente lento.
- En la plataforma, este método produce Time Limit Exceeded.

2. Método Óptimo (eficiente)

Aquí usamos la lógica matemática explicada antes:

- Si K > 1, el número repetido NUNCA será primo,
  excepto un solo caso: X = 1 y K = 2 → número 11 → primo

Entonces:

```
if (x == 1) return k == 2;
if (k > 1) return false;
```

A partir de aquí, solo necesitamos analizar X directamente.

Verificar primalidad optimizada

En vez de recorrer hasta X, solo revisamos divisores hasta su raíz usando:

```
for (long i = 2; i * i <= x; i++) {
   if (x % i == 0) return false;
}
return true;
```

La condición i * i <= x es equivalente a i <= sqrt(x)

pero evita problemas de precisión de Math.sqrt.

Complejidad O(√X)

Esto permite manejar números de hasta 10^9 fácilmente.

Ejecución de pruebas en el main

En las tres instancias:

1. X = 1, K = 2
   - Óptimo: detecta directamente que es el único caso válido.
   - Cándido: debe construir "11" y verificar primalidad.
2. X = 9, K = 7
   - Óptimo: retorna falso inmediatamente (K > 1).
   - Cándido: crea un número enorme "9999999" y tarda muchísimo.
3. X = primo grande, K = 1
   - Óptimo: solo recorre hasta √X.
   - Cándido: recorre hasta el número mismo → súper lento.

Resultado real enviado a Codeforces:

- Método Cándido → Time Limit Exceeded
- Método Óptimo → Accepted

Conclusión
El ejercicio demuestra cómo:

- Un análisis matemático adecuado evita cálculos enormes.
- La verificación de primalidad debe implementarse con método $\sqrt{N}$.
- Evitar construir números gigantes es esencial.

El método óptimo es la única solución viable para los límites del problema.