To create a compressed public key for a Bitcoin wallet in Java, you can make use of a cryptographic library like Bouncy Castle. Below is an example code snippet that demonstrates how to generate a compressed public key from a given private key using the Bouncy Castle library:

import org.bouncycastle.crypto.digests.SHA256Digest;

import org.bouncycastle.crypto.ec.CustomNamedCurves;

import org.bouncycastle.crypto.params.ECDomainParameters;

import org.bouncycastle.crypto.params.ECPrivateKeyParameters;

import org.bouncycastle.crypto.params.ECPublicKeyParameters;

import org.bouncycastle.crypto.signers.ECDSASigner;

import org.bouncycastle.math.ec.ECAlgorithms;

import org.bouncycastle.math.ec.ECCurve;

import org.bouncycastle.math.ec.ECPoint;

import java.math.BigInteger;

import java.security.SecureRandom;

public class BitcoinKeyGenerator {

public static void main(String[] args) {

// Generate a random private key

BigInteger privateKey = new BigInteger(256, new SecureRandom());

// Generate the compressed public key

ECPoint publicKey = generateCompressedPublicKey(privateKey);

// Display the compressed public key

System.out.println("Private Key: " + privateKey.toString(16));

System.out.println("Compressed Public Key: " + publicKey.getEncoded(true).toString(16));

}

private static ECPoint generateCompressedPublicKey(BigInteger privateKey) {

// Define the elliptic curve parameters for Bitcoin (secp256k1)

ECCurve curve = CustomNamedCurves.getByName("secp256k1");

ECDomainParameters domainParams = new ECDomainParameters(curve,

curve.decodePoint(hexStringToByteArray("04"

+ "FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFC2F"))));

// Create the private key parameters

ECPrivateKeyParameters privateKeyParams = new ECPrivateKeyParameters(privateKey, domainParams);

// Compute the public key from the private key

ECPoint publicKey = ECAlgorithms.GENERATE\_PUBLIC\_KEY(curve, privateKeyParams.getD());

// Compress the public key

ECPoint compressedPublicKey = publicKey.normalize().multiply(BigInteger.valueOf(2));

// Add the appropriate prefix (02 for even y, 03 for odd y)

if (compressedPublicKey.getY().testBit(0)) {

compressedPublicKey = compressedPublicKey.add(BigInteger.ONE);

}

return compressedPublicKey;

}

private static byte[] hexStringToByteArray(String hexString) {

int len = hexString.length();

byte[] data = new byte[len / 2];

for (int i = 0; i < len; i += 2) {

data[i / 2] = (byte) ((Character.digit(hexString.charAt(i), 16) << 4)

+ Character.digit(hexString.charAt(i + 1), 16));

}

return data;

}

}

In this example, we're using the Bouncy Castle library to perform elliptic curve operations. The generateCompressedPublicKey method takes a BigInteger private key as input and returns the corresponding compressed public key as an ECPoint object. The resulting compressed public key is then displayed in hexadecimal format.

Note that you need to have the Bouncy Castle library added to your Java project for this code to compile and run successfully. You can download the JAR file from the Bouncy Castle website (<https://www.bouncycastle.org/latest_releases.html>) and include it in your project's classpath.