This is another Java class named `VoterJwtService`, which is responsible for handling operations related to JSON Web Tokens (JWT) for authentication and authorization purposes. Let's go through the code and explain each part:

1. `@Service`: This annotation marks the class as a Spring service, making it eligible for automatic dependency injection and component scanning.

2. `@Value`: This annotation is used to inject values from configuration properties. The values for `secretKey`, `jwtExpiration`, and `refreshExpiration` are read from the configuration properties using the `${...}` syntax.

3. `public String extractUsername(String token)`: This method takes a JWT token as input and extracts the subject (username) from the token claims. It uses the `extractClaim` method to perform this extraction.

4. `public <T> T extractClaim(String token, Function<Claims, T> claimsResolver)`: This is a generic method that takes a JWT token and a `Function` to extract a specific claim from the token. It uses the `extractAllClaims` method to get all the claims from the token and then applies the provided `claimsResolver` function to extract the desired claim.

5. `public String generateToken(UserDetails userDetails)`: This method generates a JWT token for the provided `UserDetails` object. It internally calls the overloaded `generateToken` method with an empty map of extra claims and the specified `jwtExpiration` as the token expiration time.

6. `public String generateToken(Map<String, Object> extraClaims, UserDetails userDetails)`: This overloaded method allows the inclusion of additional claims in the generated token. It calls the `buildToken` method to construct the JWT token.

7. `private String buildToken(Map<String, Object> extraClaims, UserDetails userDetails, long expiration)`: This method builds the JWT token using the `Jwts` builder. It sets the claims, subject (username), issued and expiration dates, and signs the token using the `getSignInKey()`.

8. `public boolean isTokenValid(String token, UserDetails userDetails)`: This method checks if the provided JWT token is valid. It compares the username extracted from the token with the username from the `UserDetails` object and also checks if the token is not expired.

9. `private boolean isTokenExpired(String token)`: This private method checks if the provided JWT token is expired by comparing the expiration date extracted from the token with the current date.

10. `private Date extractExpiration(String token)`: This private method extracts the expiration date from the claims of the JWT token using the `extractClaim` method.

11. `private Claims extractAllClaims(String token)`: This private method parses the JWT token and extracts all the claims using the `Jwts.parserBuilder()` and `parseClaimsJws(token)` methods.

12. `private Key getSignInKey()`: This private method retrieves the signing key for the JWT token. It decodes the `secretKey` from Base64 and returns it as a `Key` object using `Keys.hmacShaKeyFor(keyBytes)`.

In summary, the `VoterJwtService` class provides methods to generate and validate JWT tokens for user authentication. It also allows the inclusion of additional claims in the token, making it suitable for use in various scenarios like user authorization and other custom requirements. The `secretKey`, `jwtExpiration`, and `refreshExpiration` values are read from configuration properties, making the JWT-related settings configurable.