```
'use strict';

/**
 * Module dependencies.
 */

var AbstractGrantType = require('./abstract-grant-type');
var InvalidArgumentError = require('../errors/invalid-argument-error');
var InvalidGrantError = require('../errors/invalid-grant-error');
var InvalidRequestError = require('../errors/invalid-request-error');
var Promise = require('bluebird');
var promisify = require('promisify-any').use(Promise);
var ServerError = require('../errors/server-error');
var is = require('../validator/is');
var util = require('util');

/**
 * Constructor.
 */

function AuthorizationCodeGrantType(options) {
  options = options || {};

  if (!options.model) {
    throw new InvalidArgumentError('Missing parameter: `model`');
  }

  if (!options.model.getAuthorizationCode) {
    throw new InvalidArgumentError('Invalid argument: model does not implement `getAuthorizationCode()`');
  }

  if (!options.model.revokeAuthorizationCode) {
    throw new InvalidArgumentError('Invalid argument: model does not implement `revokeAuthorizationCode()`');
  }

  if (!options.model.saveToken) {
    throw new InvalidArgumentError('Invalid argument: model does not implement `saveToken()`');
  }

  AbstractGrantType.call(this, options);
}

/**
 * Inherit prototype.
 */

util.inherits(AuthorizationCodeGrantType, AbstractGrantType);

/**
 * Handle authorization code grant.
 *
 * @see https://tools.ietf.org/html/rfc6749#section-4.1.3
 */

AuthorizationCodeGrantType.prototype.handle = function(request, client) {
  if (!request) {
    throw new InvalidArgumentError('Missing parameter: `request`');
  }

  if (!client) {
    throw new InvalidArgumentError('Missing parameter: `client`');
  }

  return Promise.bind(this)
    .then(function() {
      return this.getAuthorizationCode(request, client);
    })
    .tap(function(code) {
      return this.validateRedirectUri(request, code);
    })
    .tap(function(code) {
      return this.revokeAuthorizationCode(code);
    })
    .then(function(code) {
      return this.saveToken(code.user, client, code.authorizationCode, code.scope);
    });
};

```

```
/**
 * Get the authorization code.
 */

AuthorizationCodeGrantType.prototype.getAuthorizationCode = function(request, client) {
  if (!request.body.code) {
    throw new InvalidRequestError('Missing parameter: `code`');
  }

  if (!is.vschar(request.body.code)) {
    throw new InvalidRequestError('Invalid parameter: `code`');
  }
  return promisify(this.model.getAuthorizationCode, 1).call(this.model, request.body.code)
    .then(function(code) {
      if (!code) {
        throw new InvalidGrantError('Invalid grant: authorization code is invalid');
      }

      if (!code.client) {
        throw new ServerError('Server error: `getAuthorizationCode()` did not return a `client` object');
      }

      if (!code.user) {
        throw new ServerError('Server error: `getAuthorizationCode()` did not return a `user` object');
      }

      if (code.client.id !== client.id) {
        throw new InvalidGrantError('Invalid grant: authorization code is invalid');
      }

      if (!(code.expiresAt instanceof Date)) {
        throw new ServerError('Server error: `expiresAt` must be a Date instance');
      }

      if (code.expiresAt < new Date()) {
        throw new InvalidGrantError('Invalid grant: authorization code has expired');
      }

      if (code.redirectUri && !is.uri(code.redirectUri)) {
        throw new InvalidGrantError('Invalid grant: `redirect_uri` is not a valid URI');
      }

      return code;
    });
};

/**
 * Validate the redirect URI.
 *
 * "The authorization server MUST ensure that the redirect_uri parameter is
 * present if the redirect_uri parameter was included in the initial
 * authorization request as described in Section 4.1.1, and if included
 * ensure that their values are identical."
 *
 * @see https://tools.ietf.org/html/rfc6749#section-4.1.3
 */

AuthorizationCodeGrantType.prototype.validateRedirectUri = function(request, code) {
  if (!code.redirectUri) {
    return;
  }

  var redirectUri = request.body.redirect_uri || request.query.redirect_uri;

  if (!is.uri(redirectUri)) {
    throw new InvalidRequestError('Invalid request: `redirect_uri` is not a valid URI');
  }

  if (redirectUri !== code.redirectUri) {
    throw new InvalidRequestError('Invalid request: `redirect_uri` is invalid');
  }
};

/**
 * Revoke the authorization code.
 *
 * "The authorization code MUST expire shortly after it is issued to mitigate
 * the risk of leaks. [...] If an authorization code is used more than once,
 * the authorization server MUST deny the request."
 *
 * @see https://tools.ietf.org/html/rfc6749#section-4.1.2
 */

AuthorizationCodeGrantType.prototype.revokeAuthorizationCode = function(code) {
  return promisify(this.model.revokeAuthorizationCode, 1).call(this.model, code)
    .then(function(status) {
      if (!status) {
        throw new InvalidGrantError('Invalid grant: authorization code is invalid');
      }

      return code;
    });
};

/**
 * Save token.
 */
```

```
AuthorizationCodeGrantType.prototype.saveToken = function(user, client, authorizationCode, scope) {
  var fns = [
    this.validateScope(user, client, scope),
    this.generateAccessToken(client, user, scope),
    this.generateRefreshToken(client, user, scope),
    this.getAccessTokenExpiresAt(),
    this.getRefreshTokenExpiresAt()
  ];

  return Promise.all(fns)
    .bind(this)
    .spread(function(scope, accessToken, refreshToken, accessTokenExpiresAt, refreshTokenExpiresAt) {
      var token = {
        accessToken: accessToken,
        authorizationCode: authorizationCode,
        accessTokenExpiresAt: accessTokenExpiresAt,
        refreshToken: refreshToken,
        refreshTokenExpiresAt: refreshTokenExpiresAt,
        scope: scope
      };

      return promisify(this.model.saveToken, 3).call(this.model, token, client, user);
    });
};

/**
 * Export constructor.
 */

module.exports = AuthorizationCodeGrantType;
```