IDTF CODING REQUIREMENTS

1. Try to use short but descriptive names for all local variables. The best way is to use lower case and underscores like presented below:

   ```
   $data, $members_list, $first_name, $fname
   ```

   but you can also use another notation that is similiar to „Hungarian notation”:

   ```
   $oMember (object), $aMembers (array), $nCounter (integer), $bSuccess (boolean), $sText (string)
   ```

   it can be very helpful in the case there is long code to remember a variable type.

2. Use tabulation - 4 characters long, it's not too short and not too long, so the code looks pretty well.

3. Always try to keep the code optimized! Try to do not do as many unnecessary things as you can and always try to choose the fastest way to accomplish the job.

4. For class members (variables) always start with underscore (eg. `private $_adapter`). Do the same for private class methods.

5. We prefer a style of the code as presented below:

```
public function editprofileAction()
{
    $user = Application_Model_User::loadById( $this->getRequest()->getParam('uid') );
    $form = new DEMO_Form_UserProfile( DEMO_Form_UserProfile::EDIT_PROFILE );

    // form submitted - process...
    if ($this->getRequest()->getParam('save_profile') !== null) {

        $in = $this->getRequest()->getParams();

        if ($form->isValid($in))
        {
            $user->name = $in['name'];
            ...
        }

        if ($user->save())
        {
            $this->_helper->flashMessenger->addMessage(_T('PROFILE_UPDATED'));
            $form->populateUserData($user);
        }
        else
        {
            $form->populate($in);
            $this->_helper->flashMessenger->addMessage(_T('ENTERED_DATA_ARE_NOT_VALID'));
        }
    }
    else
    {
        $form->populateUserData($user);
    }

    $this->view->assign(array(
            'page_header' => sprintf(
                    _T('EDIT_PROFILE'),
                    $user->name.$user->surname,
                    $user->login
                    ),
            'form' => $form
        ));
}
```

6. When you're writing conditional statements ALWAYS use a space between expressions and conditions.

   ```
   if (isset($oMember) && $nMemberID == $nID)
   {
           ...
   }
   ```

7. Every constant defined should be uppercase.

8. Name of a class should always start with capital letter and all of its methods should start lowercase. It's good to define visibility to all class members.

   See below:

```
class Application_Model_Member extends Application_ModelBase
{
        public    $_table_name;
        protected $_id;
        protected $_fields = array();

        static protected function doSomething($param, $str) {
        ...
        }
}
```

9. Write useful comments when the code is long or when it seems complicated on the first look.

10. Instead of using many parameters (more than 4-5), consider to use a single variable as associative array that provide parametrized input.

11. Remember about the other programmers. The code should be relatively simple to understand and methods shouldn't be too long (or in that case - code should provide comments).

12. Code documentation. Follow the doxygen format to document the code.

```php
<?php
/*
 *  \package IDTF/Model
 *
 *  \class IDTF_Model_Abstract
 *  \brief the base class to implement by ordianry models
 *
 */
abstract class IDTF_Model_Abstract
{
    // table name (rather required when you want to use update(), delete(), create()
    protected $_table           = '';

    // list of columns that consists the PK
    protected $_primary         = array('id');

    // name of auto_increment column (needed to assign proper ID after successful create())
    protected $_autoincrement   = null;

    // default database adapter to use by model class
    protected $_adapter         = null;



    // Should return list of all available column for the table
    abstract function getColumns();

    protected function __construct()
    {
        $this->_adapter = IDTF_Model_Manager::instance()->getAdapter();
    }

    /*! \brief Select entities using plain SQL query.
     *
     *  \param $select plain SQL SELECT query
     *
     *  \return null when failed or an array of object after success (it might
     *          be empty array in the case that query returns empty result set)
     */
    protected function _select($select)
    {
        if ($r = $this->_adapter->query($select))
        {
            $out = array();
            while($row = $r->fetch())
            {
                $m = clone($this);
                $m->fromArray($row);
                $out[] = $m;
            }
            $r->free();
            return $out;
        }
        return null;
    }

    /*! \brief  Select record using only primary key.
     *
     *  After successfull execution the object that invoke that method should
     *  be filled out by all fields from the record.
     *
     *  \return true on success or false when failed.
     */
    protected function _simpleSelect($ids)
    {
```