Demonstrate an understanding of block architecture and its use in development.

View\Element\AbstractBlock:

- data automatically assigns to _data, can access data arguments later, e.g. in _construct
- `jsLayout` data argument
- toHtml:
  - event `view_block_abstract_to_html_before`
  - disable module output still works
  - if not in cache:
    - _beforeToHtml
    - _toHtml
    - save cache
  - _afterToHtml - always, even when cached
  - event `view_block_abstract_to_html_after`

View\Element\Template:

- `template` data argument
- `_viewVars` property, assign()
- `_toHtml` renders template when defined
  - getTemplateFile = View\Element\Template\File\Resolver::getTemplateFileName
  - fetchView
    - View\Element\Template\File\Validator::isValid - checks allowed directory (view_preprocessed/module/theme) or symlink
    - if bad file, log and throw error in DEV mode
    - View\TemplateEnginePool::get by file extension
      - phtml - View\TemplateEngine\Php - extract vars, include file
      - xhtml - View\TemplateEngine\Xhtml
      - \Magento\Developer\Model\TemplateEngine\Decorator\DebugHints
    - engine.render(block, template, viewVars)
- default `getCacheKeyInfo` - store code, template, base URL. By default*store aware*
- helper methods:
  - getMediaDirectory
  - getRootDirectory
  - getObjectData
  - getBaseUrl

admin configuration `dev/template/allow_symlink`, default false

Which objects are accessible from the block?

- $this = \Magento\Framework\View\TemplateEngine\Php
- $this->something() is proxied to $block->something() via magic `__call` function - only public available!
- isset($this->something) -> isset($block->something)
- $this->property = $block->property
- $this->helper() gets singleton of AbstractHelper

What is the typical block's role?

As much business logic as possible should be moved out of template, and blocks provide access to processed data for templates. Actual data crunching can (and should) be proxied further to models.

Identify the stages in the lifecycle of a block.

- block is generated
    - builder.generateLayoutBlocks
    - layout.generateElements
    - readerPool.interpret - View\Layout\Reader\Block::interpret - resolves params, schedules structure element
        - `<action ifconfig method>`
        - `<arguments>`
        - attributes: class, group, template, ttl, display, acl
        - visibilityConditions: attribute `ifconfig`, attribuet `aclResource`, child `<visibilityCondition name="" className=""><arguments.../></visibilityCondition>`
    - generatorPool.process - View\Layout\Generator\Block::process - actually creates block
        - generate all blocks `__construct`, `_construct`
        - set layout to all blocks, event after each `block.setLayout`, `block._prepareLayout` event `core_layout_block_create_after`
        - call all actions

generator.generateBlock:

- generator.createBlock -> `__construct()` -> `_construct` -> check argument ‘template’ -> setTemplate
    - generator.getBlockInstance
    - block.setType(classname)
    - block.setNameInLayout
    - block.addData(arguments.data)
- block.setTemplate (when available)
- block.setTtl (when available)

Arguments

Block arguments are treated like data-arguments. Arguments values set in a layout file can be accessed in templates using the get{ArgumentName}() and has{ArgumentName}() methods.

Arguments values set in a layout file can be accessed in templates using the get{ArgumentName}() and has{ArgumentName}() methods. The latter returns a boolean defining whether there’s any value set. {ArgumentName} is obtained from the name attribute the following way: for getting the value of `<argument name="some_string">` the method name is getSomeString().

Magento docs - Layout instructions

Block viewModel concept

Magento 2.2 suggests moving block business logic to separate viewModel classes.

Avoid extending this template class, because with inheriting the template block is the constructor is very large.

If you need custom presentation logic in your blocks, use this class as block, and declare custom view models in block arguments in layout handle file.

View model parameter can have any name. Also, one block can have multiple view models injected via layout.

Example:

```
<block name="my.block" class="Magento\Backend\Block\Template"
template="My_Module::template.phtml" >
     <arguments>
         <argument name="viewModel"
xsi:type="object">My\Module\ViewModel\Custom</argument>
     </arguments>
</block>
```

In template, access your model instead of the block:

```
$viewModel = $block->getData('viewModel');
// or
$viewModel = $block->getViewModel();
```

ViewModels in Magento 2

In what cases would you put your code in the _prepareLayout(), _beforeToHtml(), and _toHtml() methods?

`_prepareLayout` - most commonly extended:

At this stage all blocks in layout are generated, but _prepareLayout is only running, so some blocks might still be not initialized completely.

- set page config title

  ```
  $this->pageConfig->getTitle()->set(__('Address
  Book'));
  ```

- edit head block, e.g. add RSS

- add breadcrumbs

- create new block, set as child - e.g. grid

`_beforeToHtml` :

- can’t change page title?
- some blocks are rendered, can’t change them
- assign additional template values
- delay computation to the latest point until render. If block is not rendered we save computation

`_toHtml` :

- block without template, put custom rendering here, e.g. calling external API
- suppress template output if should not render by condition - return empty string

How would you use events fired in the abstract block?

`view_block_abstract_to_html_before` :

- edit cache params - lifetime, ttl, tags
- add column to grid
- edit template params - e.g. set disable edit flag

`view_block_abstract_to_html_after` :

- edit html - replace, add content, add wrappers

Describe how blocks are rendered and cached.

toHtml, load cache, [when not in cache: _beforeToHtml, _toHtml], _afterToHtml, save cache

- data attribute `cache_lifetime` must be set
- cache key one of:
  - data attribute `cache_key` = BLOCK_ `$cache_key`
  - default getCacheKeyInfo: [name in layout]
  - or BLOCK_ `getCacheKeyInfo()` - by default name in layout. By default*same for all stores*!
- cache tags for automatic invalidation:
  - data attribute `cache_tags`
  -
    - implicit tag ‘block_html’
  - if instanceof DataObject\IdentityInterface, `getIdentities`

Identify the uses of different types of blocks.

- View\Element\AbstractBlock - custom logic
- View\Element\Template - any template
- View\Element\Text - set text programmatically
- etc.

When would you use non-template block types?

Something like CMS page - admin controls block content and layout. Another example - dynamic robots.txt whose content is stored in DB.

In what situation should you use a template block or other block types?

Use template block whenever possible to allow for theme markup customization.