Describe price functionality

Identify the basic concepts of price generation in Magento.

Terms:

- price info factory - creates *price info* and passes new *price collection* for specific product instance and qty
- *price info* - holds *adjustment collection* and *price collection*. same across all product types
- *price collection* - holds price pool
- *price pool* - holds price interfaces for every type of price - regular, special, final etc. Class is same, but a lot of copies are created by di configuration.
- *adjustment collection* - holds *adjustment pool* and adjustment codes - `tax` , `wee` , `wee_tax`
- *adjustment pool* - holds adjustments by code - `tax` sort 20, `wee` sort 25, `wee_tax` sort 35
- *price interface* - holds float value in display currency, price amount obj., can return custom amount without specific adjustments
- *amount interface* - in display currency. base, full amount, adjustments

What's `wee_tax` :

- fixed product tax (FPT) can be configured to be taxed
- tax percentage over fixed tax

Flow:

- product implements Pricing\SaleableInterface
- product.getPriceInfo
- \Magento\Catalog\Model\Product\Type::getPriceInfo
- Pricing\PriceInfo\Factory::create - by product instance and qty
    - di injectable type configurations: product_type_id => [infoClass, prices]
    - creates *price collection* = [$type_id]['prices']
        - separate *copy* of `Pricing\Price\Collection` with according *copy* of `Pricing\Price\Pool` with separate set of price info classes by type - regular, final etc.
        - ALL product types have personal collection and pool instance
    - creates *price info* = `Pricing\PriceInfo\Base`
        - MOST product types use SAME instance `Pricing\PriceInfo\Base` with default `Pricing\Adjustment\Collection`
        - Only bundle product holds custom *copy* with custom instance of `Adjustment\Collection` - but has NO changes. So it's just a separate copy.
- Pricing\PriceInfoInterface
- Pricing\Price\Collection - separate pool by product type
- Pricing\Price\Pool - iterate/access by code prices. A lot of pools per product type. `target` argument means inherit prices from other type
    - Magento\Catalog\Pricing\Price\Pool
    - Magento\Bundle\Pricing\Price\Pool

    - Magento\ConfigurableProduct\Pricing\Price\Pool
    - Magento\Downloadable\Pricing\Price\Pool
    - Magento\GroupedProduct\Pricing\Price\Pool
    - Magento\GiftCard\Pricing\Price\Pool

product type -> price collection -> price pool -> prices:

- `default` -> Magento\Catalog\Pricing\Price\Collection -> Magento\Catalog\Pricing\Price\Pool:
    - regular_price = Magento\Catalog\Pricing\Price\RegularPrice
    - final_price = Magento\Catalog\Pricing\Price\FinalPrice
    - tier_price = Magento\Catalog\Pricing\Price\TierPrice
    - special_price = Magento\Catalog\Pricing\Price\SpecialPrice
    - base_price = Magento\Catalog\Pricing\Price\BasePrice
    - custom_option_price = Magento\Catalog\Pricing\Price\CustomOptionPrice
    - configured_price = Magento\Catalog\Pricing\Price\ConfiguredPrice
    - wishlist_configured_price = Magento\Catalog\Pricing\Price\ConfiguredPrice
    - catalog_rule_price = Magento\CatalogRule\Pricing\Price\CatalogRulePrice
    - (staging) catalog_rule_price = Magento\CatalogRuleStaging\Pricing\Price\CatalogRulePrice
- `downloadable` -> Magento\Downloadable\Pricing\Price\Collection -> Magento\Downloadable\Pricing\Price\Pool:
    - link_price = Magento\Downloadable\Pricing\Price\LinkPrice
    - wishlist_configured_price = Magento\Wishlist\Pricing\ConfiguredPrice\Downloadable
    - *inherit rest from* `default`
- `configurable` -> Magento\ConfigurableProduct\Pricing\Price\Collection -> Magento\ConfigurableProduct\Pricing\Price\Pool:
    - regular_price = Magento\ConfigurableProduct\Pricing\Price\ConfigurableRegularPrice
    - final_price = Magento\ConfigurableProduct\Pricing\Price\FinalPrice
    - wishlist_configured_price = Magento\Wishlist\Pricing\ConfiguredPrice\ConfigurableProduct
    - *inherit rest from* `default`
- `bundle` -> Magento\Bundle\Pricing\Price\Collection -> Magento\Bundle\Pricing\Price\Pool:
    - regular_price = Magento\Bundle\Pricing\Price\BundleRegularPrice
    - final_price = Magento\Bundle\Pricing\Price\FinalPrice
    - tier_price = Magento\Bundle\Pricing\Price\TierPrice
    - special_price = Magento\Bundle\Pricing\Price\SpecialPrice
    - custom_option_price = Magento\Catalog\Pricing\Price\CustomOptionPrice
    - base_price = Magento\Catalog\Pricing\Price\BasePrice
    - configured_price = Magento\Bundle\Pricing\Price\ConfiguredPrice
    - bundle_option = Magento\Bundle\Pricing\Price\BundleOptionPrice
    - catalog_rule_price = Magento\CatalogRule\Pricing\Price\CatalogRulePrice
    - wishlist_configured_price = Magento\Bundle\Pricing\Price\ConfiguredPrice
- `grouped` => Magento\GroupedProduct\Pricing\Price\Collection -> Magento\GroupedProduct\Pricing\Price\Pool:
    - final_price = Magento\GroupedProduct\Pricing\Price\FinalPrice
    - configured_price = Magento\GroupedProduct\Pricing\Price\ConfiguredPrice
    - wishlist_configured_price = Magento\GroupedProduct\Pricing\Price\ConfiguredPrice

    - *inherit rest from* `default`
- `giftcard` => Magento\GiftCard\Pricing\Price\Collection => Magento\GiftCard\Pricing\Price\Pool:
    - regular_price = Magento\Catalog\Pricing\Price\RegularPrice
    - final_price = Magento\GiftCard\Pricing\Price\FinalPrice
    - tier_price = Magento\Catalog\Pricing\Price\TierPrice
    - special_price = Magento\Catalog\Pricing\Price\SpecialPrice
    - msrp_price = Magento\Msrp\Pricing\Price\MsrpPrice
    - custom_option_price = Magento\Catalog\Pricing\Price\CustomOptionPrice
    - base_price = Magento\Catalog\Pricing\Price\BasePrice
    - configured_price = Magento\GiftCard\Pricing\Price\ConfiguredPrice
    - bundle_option = Magento\Bundle\Pricing\Price\BundleOptionPrice
    - wishlist_configured_price = Magento\GiftCard\Pricing\Price\ConfiguredPrice

Default *regular price* example:

- getValue - *product.getPrice*, convert to display currency. This is raw float value without adjustments
- getAmount = Pricing\Adjustment\Calculator::getAmount:
    - priceInfo.getAdjustments
    - adjustment can be included in base price, included in display price
    - if included in base price:
        - extractAdjustment - get base price without adjustment.
        - applyAdjustment - add adjustment back
    - if included in display price:
        - applyAdjustment - add adjustment over base price

Default *final price* example:

- `final_price` delegates calculation to `base_price`
- gets all prices info, checks implementing `Pricing\Price\BasePriceProviderInterface`
    - regular_price, catalog_rule_price, special_price, tier_price
- takes min value

And so on. Most classes customize PriceInterface `getValue`, while `getAmount` and adjustment adding is the same.

Adjustment:

- isIncludedInBasePrice
    - tax = true if admin prices are including tax
    - weee = false, they are always added on top of base price
- isIncludedInDisplayPrice
    - tax = display prices incl. tax
- *extractAdjustment* - return base price without adjustment (taxes)
- *applyAdjustment* - base price + adjustment
- getSortOrder

How would you identify what is composing the final price of the product?

`default` final price:

- `final_price` delegates calculation to `base_price`

- gets all prices info, checks implementing `Pricing\Price\BasePriceProviderInterface`
  - regular_price, catalog_rule_price, special_price, tier_price
- takes min value

`configurable` final price:

- price resolver interface \Magento\ConfigurableProduct\Pricing\Price\PriceResolverInterface
- ConfigurableFinalPriceResolver -> Magento\ConfigurableProduct\Pricing\Price\ConfigurablePriceResolver
- gets products with lowest prices by *lowest price options provider*, one lowest product from each type:
  - base price
  - tier price
  - index price
  - catalog rule price
- when selecting, lowest product is also processed by *select processors*:
  - status enabled
  - assigned website match
  - in stock (`cataloginventory_stock_status`) – when hide out of stock
  - in stock – when show out of stock (?)
- get final price of each individual lowest product
- configurable final price is lowest of their values

`grouped` final price:

- lowest final price of all individual products in group

`bundle` final price:

- same as `default` final price (min of classes implementing interface `BasePriceProviderInterface`) plus `bundle_option` price - check price type fixed/dynamic, all required salable options * qty

`giftcard` final price:

- first giftcard amount

How can you customize the price calculation process?

1. Create price adjustment:

- Implements Pricing\Adjustment\AdjustmentInterface

- DI configuration:

```
<type name="Magento\Framework\Pricing\Adjustment\Collection">
    <arguments>
        <argument name="adjustments" xsi:type="array">
            <item name="somecode"
xsi:type="const">Custom\Module\Pricing\Adjustment::ADJUSTMENT_CODE</item>
        </argument>
    </arguments>
</type>
```

```
<type name="Magento\Framework\Pricing\Adjustment\Pool">
    <arguments>
        <argument name="adjustments" xsi:type="array">
            <item name="somecode" xsi:type="array">
                <item name="className"
xsi:type="string">Custom\Module\Pricing\Adjustment</item>
                <item name="sortOrder" xsi:type="string">40</item>
            </item>
        </argument>
    </arguments>
</type>
```

- implement isIncludedInBasePrice, isIncludedinDisplayPrice, extractAdjustment, applyAdjustment

1. Register some new price type with `default` price pool, implement interface `Pricing\Price\BasePriceProviderInterface`. My new price can influence final price if returns lowest price.
2. Create plugin over necessary native price classes, e.g. around getValue
3. Replace price class via DI for specific virtual pool, e.g.

```
<virtualType name="Magento\Bundle\Pricing\Price\Pool">
    <arguments>
        <argument name="prices" xsi:type="array">
            <item name="regular_price"
xsi:type="string">Custom\Module\Pricing\Price\BundleRegularPrice</item>
        </argument>
    <arguments>
</virtualType>
```

1. Replace price class via DI preference for specific class.

Describe how price is rendered in Magento

Terms:

- *global render* - shared block `Magento\Framework\Pricing\Render` declared in `default.xml` layout. Data-arguments contain global defaults.

- *local render* - specific instance block `Magento\Catalog\Pricing\Render` to render specific price in designated location. Can be multiple on the same page. Its block arguments overwrite arguments from *global renderer*. At least must set argument `price_type_code`.

- *price layout* - simple wrapper that holds separate layout instance and renders only one layout handle `catalog_product_prices`.

- *renderer pool* - shared block `Pricing\Render\RendererPool` defined in `catalog_product_prices.xml`. Data-arguments describe price box *class*, *template*, *amount renderer* and *adjustment render* for each combination of *product type* and *price type*, or defaults when not customized.

- *price box* - `Pricing\Render\PriceBoxInterface` - renders template, can render multiple prices like old price and special price

- *amount render* - `Pricing\Render\Amount` - renders one specific price inside price box - *display value* and *adjustments html*

- *adjustment render* - `Pricing\Render\AbstractAdjustment` - whatever custom adjustment markup. has access to amount render and amount

- include *local render* block in layout:

```
<block class="Magento\Catalog\Pricing\Render" name="product.price.final">
    <arguments>
        <argument name="price_render"
xsi:type="string">product.price.render.default</argument>
        <argument name="price_type_code" xsi:type="string">final_price</argument>
        <argument name="zone" xsi:type="string">item_view</argument>
        <argument name="display_msrp_help_message" xsi:type="string">1</argument>
        <argument name="id_suffix_some" xsi:type="string">copy-</argument>
    </arguments>
</block>
```

- gets layout block by name in `price_render`
- *global render* - block `product.price.render.default` = Framework\Pricing\Render is declared in `default` handle with *default arguments* - present for all pages.

```
<block class="Magento\Framework\Pricing\Render"
name="product.price.render.default">
    <arguments>
        <argument name="price_render_handle"
xsi:type="string">catalog_product_prices</argument>
        <argument name="use_link_for_as_low_as" xsi:type="boolean">true</argument>
        <!-- set "override" configuration settings here -->
    </arguments>
</block>
```

- *global render*.`_prepareLayout` :
  - has *price layout* - mini-copy of layout. it had only one block*renderer pool*, every product type add own arguments by type.
  - priceLayout.addHandle from arguments `price_render_handle`, by default `catalog_product_prices`
  - priceLayout.loadLayout - full layout processing - load, generateXml, generateElements.
- *local render*._toHtml - could be cached
- *local arguments* are merged with default arguments, additional argument `render_block` = *local render*
- *global render*.render
- get *renderer pool* from *price layout* (`catalog_product_prices` handle):
- Pricing\Render\RendererPool::createPriceRender by *product type* and requested *price code*: Searches `renderer_class` in data arguments by patterns:
  - `$type/prices/$priceCode/render_class`
  - `$type/default_render_class`
  - `default/prices/$priceCode/render_class`
  - `default/default_render_class`
- creates new found *price box* block instance of `Pricing\Render\PriceBoxRenderInterface`
- finds and sets template from data arguments by patterns:
  - `$type/prices/$priceCode/render_template`
  - `$type/default_render_template`
  - `default/prices/$priceCode/render_template`
  - `default/default_render_template`
- all *price boxes* are cached:
  `store-template-baseUrl-template-[price_id_prefix.productId.price_id_suffix]-priceCode`

E.g. `simple` product type renders `final_price` price box:

- price box class `Magento\Catalog\Pricing\Render\FinalPriceBox`
- template `Magento_Catalog::product/price/final_price.phtml`
- if has special price, shows 2 prices - '.special-price' and '.old-price':

  has special = regular price (product.getPrice) < final price (lowest of regular_price, catalog_rule_price, special_price, tier_price)
- shows as low as if applicable
- amount = finalPriceInfo.getAmount() = price.getValue + adjustments in display currency
- price box.renderAmount(amount)
- price box.getAmountRender
- renderer pool.createAmountRender - from data-arguments by patterns:
  - `$type/prices/$priceCode/amount_render_class`
  - `$type/default_amount_render_class`
  - `default/prices/$priceCode/amount_render_class`
  - `default/default_amount_render_class`
- amount render template:
  - `$type/prices/$priceCode/amount_render_template`
  - `$type/default_amount_render_template`
  - `default/prices/$priceCode/amount_render_template`
  - `default/default_amount_render_template`

E.g. `simple` product renders `final_price` amount:

- amount render class `Pricing\Render\Amount`
- amount render template `Magento_Catalog::product/price/amount/default.phtml` - '.price-container'
- amount render.getDisplayValue = amount.getValue() = price.getValue() + adjustments
- amount render.getAdjustmentsHtml
  - renderer pool.getAdjustmentRenders - finds adjustment = [class, template] by patterns - can be multiple!
    - `$type/adjustments/$priceCode`
    - `$type/adjustments/default`
    - `default/adjustments/$priceCode`
    - `default/adjustments/default` E.g. \Magento\Tax\Pricing\Render\Adjustment with `Magento_Tax::pricing/adjustment.phtml`
- amount render.getAdjustments(array)
  - can suppress adjustments by setting data `skip_adjustments`
  - set css classes from adjustment codes
  - check that adjustment applies: amount.getAdjustmentAmount(code)
  - adjustment render.render
  - adjustment render.apply - must implement abstract method, e.g. call `$this->toHtml`. Can set back data-attributes to amount render, e.g. classes and labels.

How would you render price in a given place on the page

Include new copy block type `Magento\Catalog\Pricing\Render`, pass required value for `price_type_code` argument.

… and how would you modify how the price is rendered?

- use data-arguments to customize data for *price box*, *amount renders* and *adjustment renders*:
  - css_classes, price_id, price_id_prefix, price_id_suffix
- create layout handle `catalog_product_prices.xml`, reference block `render.product.prices` and customize arguments.
  - (product type/price type or default type) *price box* class and template, *amount render* class and template
  - (product type/price type or default type) specific adjustment class and template