Objects and Classes

The Big Picture

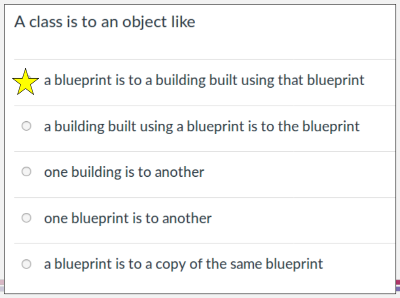
With subroutines, we have started thinking about program organization.

Objects are the next step in organizing program and building modules –

- we can visually group statements that together perform a task by putting blank lines before and after
- we can group statements that together perform a single whole task into a subroutine, then treat that subroutine like a black box
- we can group subroutines and variables that together have a single whole purpose into an object, and treat that object like a bigger black box
  - in doing this, we define new types

Classes and Objects – Concepts

- classes in Java typically have one of two purposes
  - (so far) as holders for `static` subroutines, possibly including `main`
  - (new) as templates for constructing objects
- objects are *instances* of a class



A class is to an object like

[✓ a blueprint is to a building built using that blueprint]

○ a building built using a blueprint is to the blueprint

○ one building is to another

○ one blueprint is to another

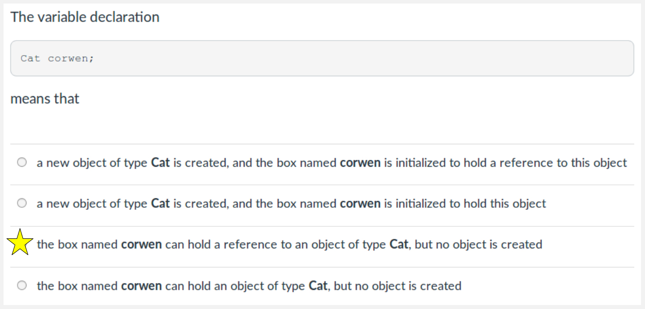
○ a blueprint is to a copy of the same blueprint

Using Objects

Three steps –

- declare a variable to hold a reference to the object
- create the object itself (using a *constructor*)
- use the object by invoking methods on it

Find out how to use the constructor and what methods are available using the API documentation for the class.



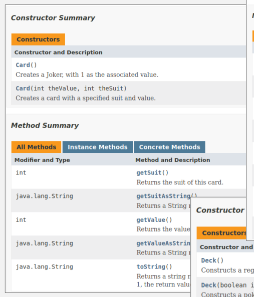
The variable declaration

```
Cat corwen;
```

means that

- ○ a new object of type **Cat** is created, and the box named **corwen** is initialized to hold a reference to this object
- ○ a new object of type **Cat** is created, and the box named **corwen** is initialized to hold this object
- [✓ the box named **corwen** can hold a reference to an object of type **Cat**, but no object is created]
- ○ the box named **corwen** can hold an object of type **Cat**, but no object is created

Reading APIs



Constructor Summary

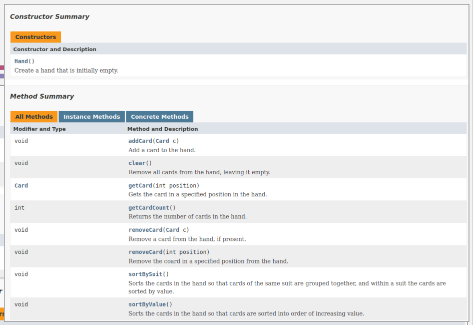
Constructors

Constructor and Description
`Card()` Creates a Joker, with 1 as the associated value.
`Card(int theValue, int theSuit)` Creates a card with a specified suit and value.

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
`int`	`getSuit()` Returns the suit of this card.
`java.lang.String`	`getSuitAsString()` Returns a String r
`int`	`getValue()` Returns the value
`java.lang.String`	`getValueAsStrin` Returns a String r
`java.lang.String`	`toString()` Returns a string r 1, the return valu



Constructor Summary

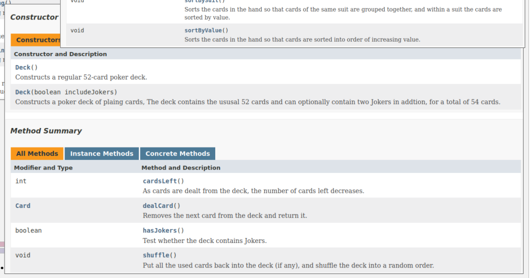
Constructors

Constructor and Description
`Hand()` Create a hand that is initially empty.

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
`void`	`addCard(Card c)` Add a card to the hand.
`void`	`clear()` Remove all cards from the hand, leaving it empty.
`Card`	`getCard(int position)` Gets the card in a specified position in the hand.
`int`	`getCardCount()` Returns the number of cards in the hand.
`void`	`removeCard(Card c)` Remove a card from the hand, if present.
`void`	`removeCard(int position)` Remove the coard in a specified position from the hand.
`void`	`sortBySuit()` Sorts the cards in the hand so that cards of the same suit are grouped together, and within a suit the cards are sorted by value.
`void`	`sortByValue()` Sorts the cards in the hand so that cards are sorted into order of increasing value.



Constructor

Constructors

Constructor and Description
`Deck()` Constructs a regular 52-card poker deck.
`Deck(boolean includeJokers)` Constructs a poker deck of plaing cards, The deck contains the ususal 52 cards and can optionally contain two Jokers in addtion, for a total of 54 cards.

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
`int`	`cardsLeft()` As cards are dealt from the deck, the number of cards left decreases.
`Card`	`dealCard()` Removes the next card from the deck and return it.
`boolean`	`hasJokers()` Test whether the deck contains Jokers.
`void`	`shuffle()` Put all the used cards back into the deck (if any), and shuffle the deck into a random order.