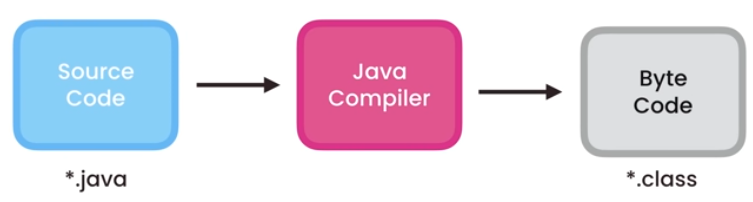
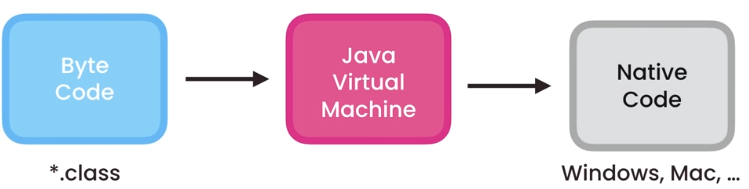
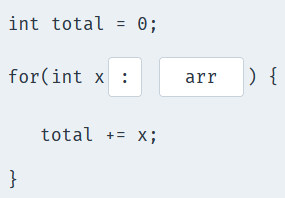
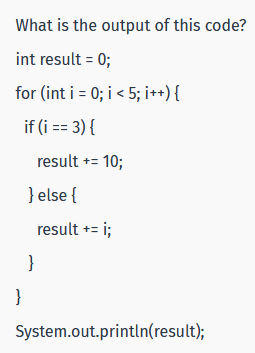
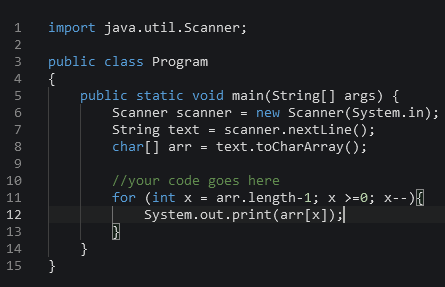
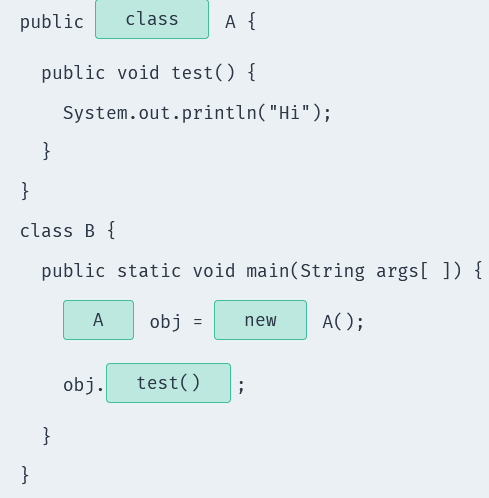
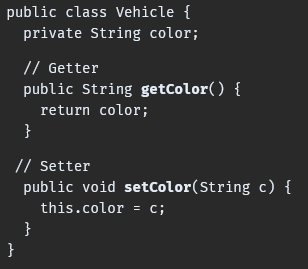
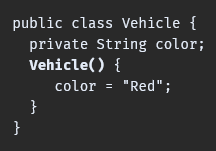
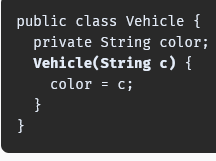
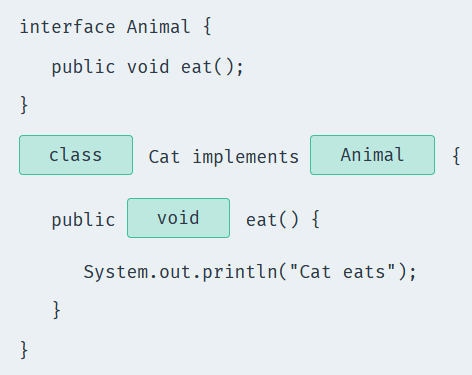
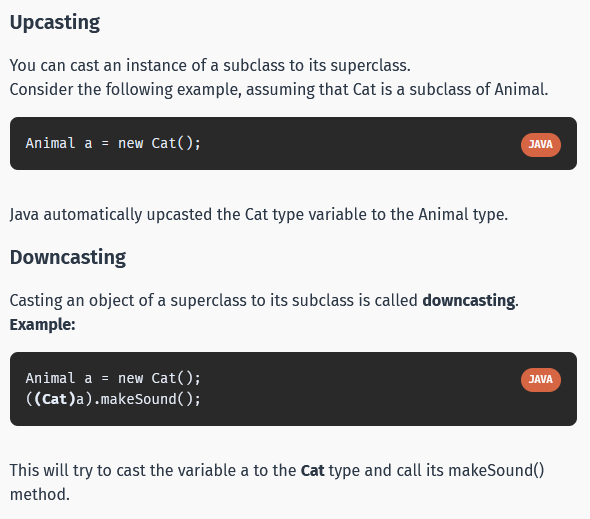
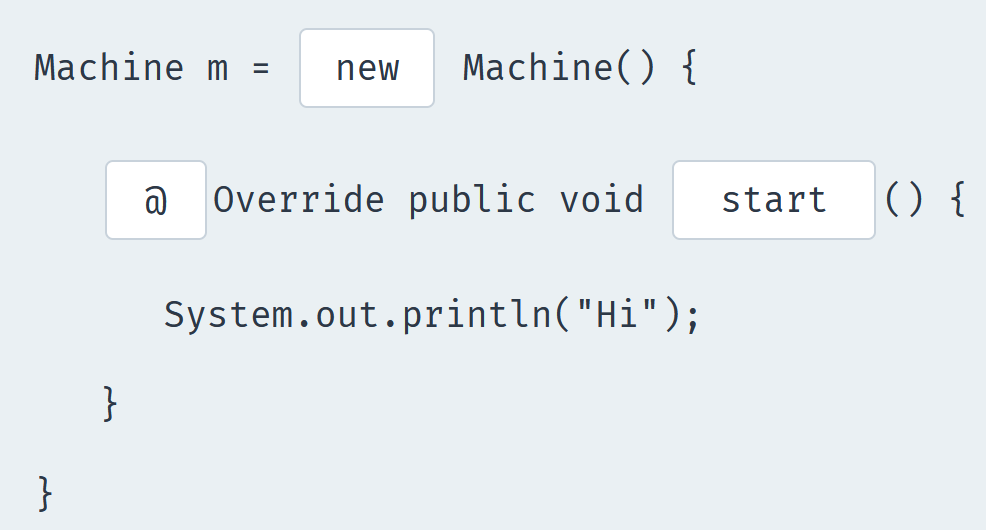
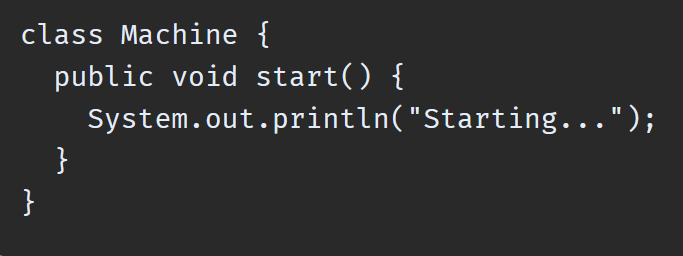
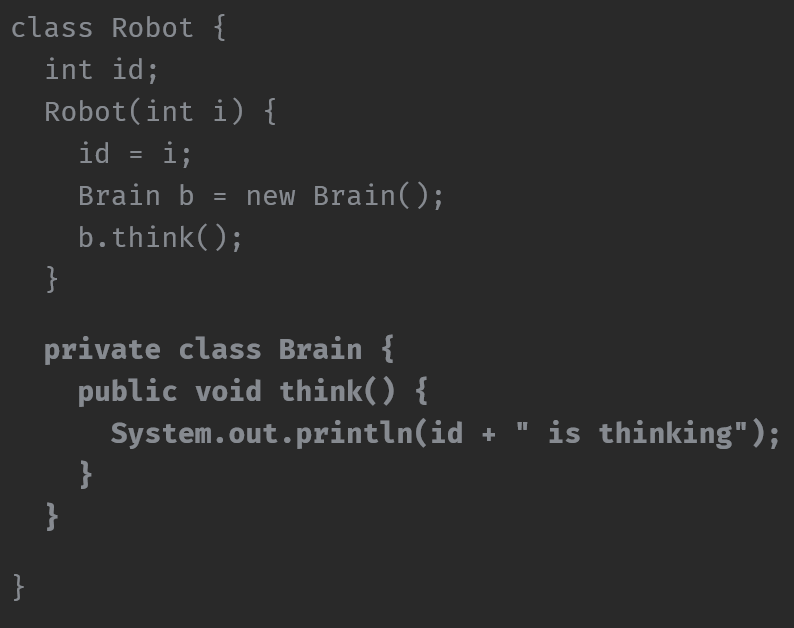
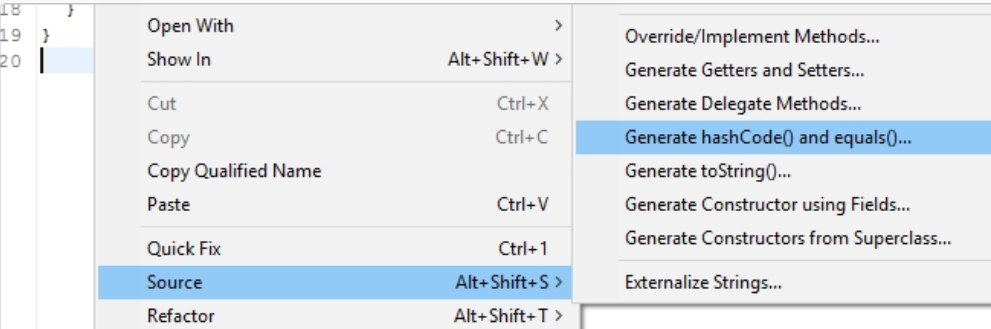
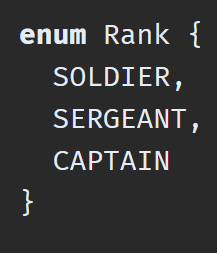
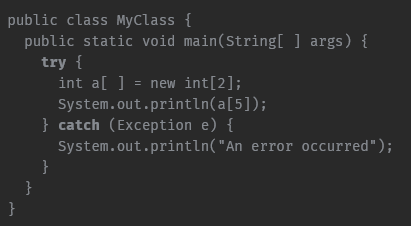
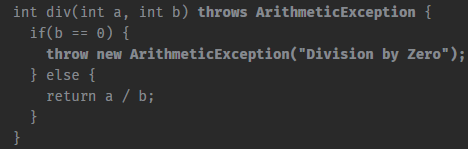
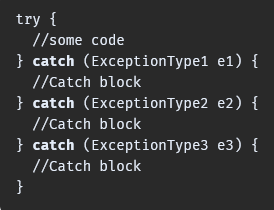
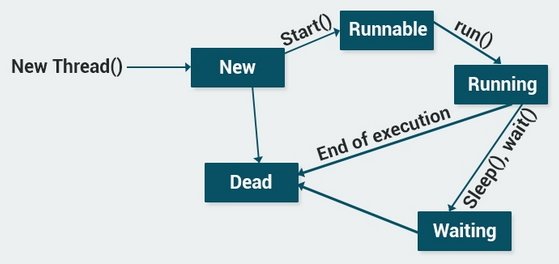
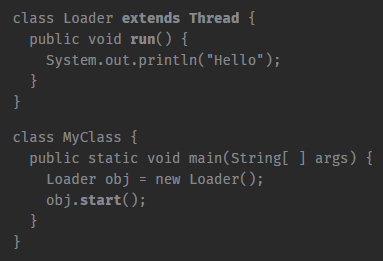
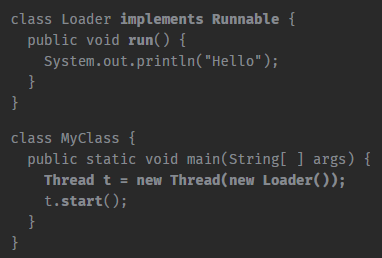
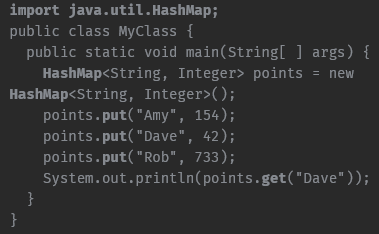
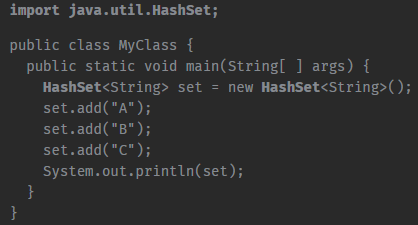
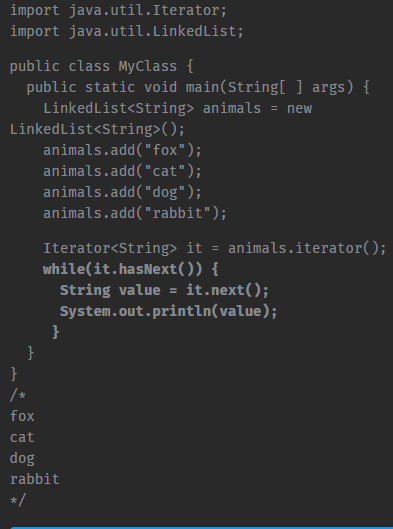
* JDK > JRE > JVM
* Compilation
  + 
* JRE Java Runtime Environment
  + 
* Prefix and Postfix for Increment/Decrement
  + Prefix – increments the variable’s value and uses the new value in the expression
    - ++a
  + Postfix – the variable’s value is first used in the expression and is then increased
    - a++
* Escape sequence
  + “Hello \”World\””
    - Allow you to print the double quotes
* Constant
  + final float PI = 3.14F;
    - use keyword “final” to signify constant
    - Naming convention for constant is all cap
* Implicit casting
  + byte > short > int > long > float > double
* Explicit casting
  + double x = 1.1;
  + int y = (int)x + 2;
* Each loop – enhanced for loop
  + Ex: to calculate the sum of an array
    - 
* What is the output of this code? – if statement within for loop
  + 
  + Output: 17
* Reversing a string
  + 
* Object is an instance of a class
* Create an object of the A class in the B class and call its “test” method
  + 
* Creating a new object of a class
  + People obj1 = new People();
* Access modifier
  + public – the class is accessible by any other class
  + default – the class is accessible only by classes in the same package
  + protected – provides the same access as the default access modifier, with the addition that subclasses can access protected methods and variables of the superclass
  + private – accessible only within the declared class itself
* Getters and Setters
  + Used to protect your data, particularly when creating classes
  + The getter method returns the value of the attribute
  + The setter method takes a parameter and assigns it to the attribute
  + Ex: 
* Constructors – special methods invoked when an object is created and are used to initialize them
  + A constructor can be used to provide initial values for object attributes
  + A constructor name must be same as its class name
  + A constructor must have no explicit return type
  + Ex: 
    - The Vehicle() method is the constructor of our class, so whenever an object of that class is created, the color attribute will be set to “Red”. A constructor can also take parameters to initialize attributes
    - 
  + You can think of constructors as methods that will set up your class by defaults, so you don’t need to repeat the same code every time
* Math class
  + Math.abs()
  + Math.ceil() – round up
  + Math.floor() – round down
  + Math.pow() – takes two parameters and returns the first parameter raised to the power of the second parameter
  + Math.min() and Math.max()
* Static – when you declare a variable or a method as static, it belongs to the class, rather than to a specific instance. This means that only one instance of a static member exists, even if you create multiple objects of the class, or if you don’t create any. It will be shared by all objects
  + Another example of static methods are those of the Math class, which is why you can call them without creating a Math object
* Encapsulation – details are not visible to users; data hiding
  + Declare the class variables as private and provide public setter and getter
* Inheritance – enables one class to acquire the properties (methods and variables) of another
  + Subclass (a.k.a. derived class, child class) – the class inheriting the properties of another
    - When one class is inherited from another class, it inherits all of the superclass’ non-private variables and methods
  + Superclass – the class whose properties are inherited
  + Use “extends” keyword
  + Constructors are not member methods, and so are not inherited by subclass
    - However, the constructor of the superclass is called when the subclass is instantiated
  + You can access the superclass from the subclass using the “super” keyword
* Polymorphism – having different implementation of a method
  + Achieved through overriding and overloading
    - Overriding – writing over a method
      * Should have the same return type and arguments
      * The access level cannot be more restrictive
      * A method declared final or static cannot be overridden
      * If a method cannot be inherited, it cannot be overridden
      * Constructors cannot be overridden
      * Methods overriding is also known as runtime polymorphism
    - Overloading – when methods have the same name, but different parameters
      * Another name for method overloading is compile-time polymorphism
* Abstraction – hiding implementation
  + Achieved using abstract classes and interfaces
  + An abstract class is defined using the abstract keyword
  + If a class is declared abstract it cannot be instantiated (you cannot create objects of that type)
  + To use an abstract class, you have to inherit it from another class
  + Any class that contains an abstract method should be defined as abstract
  + An abstract method is a method that is declared without an implementation (without braces, and followed by a semicolon):
    - abstract void walk();
* Interfaces – a completely abstract class that only contains abstract methods
  + Defined using the “interface” keyword
  + Use the “implement” keyword to use an interface with your class
  + May contain only static final variables
  + Cannot contain a constructor because interfaces cannot be instantiated
  + A class can inherit from just one superclass, but can implement multiple interfaces
  + A class can implement any number of interfaces
  + Ex: 
  + Methods in an interface are implicitly public
* Typecasting class
  + 
  + Widening – the data is implicitly casted from a smaller primitive type to a larger primitive type. A.k.a. auto-widening. Ex: byte to short, short to int, int to long, and float to double
  + Narrowing – you must explicitly cast the data from a larger primitive type to a smaller primitive type. A.k.a. explicit narrowing
  + Derived casting – casting from one object to another object
  + Class cast exception – an exception which occurs at runtime when an object of one type cannot be casted to another type
  + Boxing – wrapping a primitive type into the corresponding wrapper class
  + Unboxing – when the runtime retrieves the primitive datatype from the wrapper class
* @Override
  + Ex: 
    - The @Override annotation is used to make it more obvious when methods are overridden
    - Override previous method: 
* Inner class – nesting class, basically just a class inside another class. Unlike a class, an inner class can be private
  + 
* hashCode() and equals()
  + Each object has a predefined equal() method that is used for semantical equality testing. But, to make it work for our classes, we need to override it and check the conditions we need
    - 
  + The automatically generated hashCode() method is used to determine where to store object internally. Whenever you implement equals, you MUST also implement hashCode
* Enums – a special type used to define collections of constants
  + Ex: 
  + After declaring an Enum, we can check for the corresponding values with, for example, a switch statement
* Exceptions – a problem that occurs during program execution which cause abnormal termination of the program
  + Exception handling is a powerful mechanism that handles runtime errors to maintain normal application flow
  + Catch statement – if an exception occurs in the try block, the catch block that follows the try is checked. If the type of exception that occurred is listed in a catch block, the exception is passed to the catch block much as an argument is passed into a method parameter
    - Ex: 
      * The (Exception e) statement in the catch block is used to catch all possible Exceptions
      * The example demonstrate exception handling when trying to access an array index that does not exist
* Throw – the throw keyword allows you to manually generate exception from your methods
  + Ex: 
  + The throws statement in the method definition defines the type of Exceptions(s) the method can throw
  + Multiple exceptions can be dfined in the throws statement using a comma-separated list
* Multiple Exceptions
  + 
* Thread
  + Java is a multi-threaded programming language. This means that our program can make optimal use of available resources by running two or more components concurrently, with each component handling a different task
    - You can subdivide specific operations within a single application into individual threads that all run in parallel
  + Life cycle of a thread
    - 
  + 2 ways to create a thread:
    - 1. Extend the thread class
      * Inherit from the Thread class, override its run() method, and write the functionality of the thread in the run() method
      * Then you crate a new object of your class and call its start method to run the thread
      * 
        + As you can see, our Loader class extends the Thread class and overrides its run() method
        + When we create the obj object and call its start() method, the run() method statement execute on a different thread
    - 2. Implementing the Runnable interface
      * Implement the run() method, then create a new Thread object, pass the Runnable class to its constructor, and start the Thread by calling the start() method
      * 
      * It may seem that implementing the Runnable interface is a bit more complex than extending from the Thread class. However, implementing the Runnable interface is the preferred way to start a Thread, because it enables you to extend from another class, as well
    - Every Java thread is prioritized to help the operating system determine the order in which to schedule threads. The priorities range from 1 to 10, with each thread defaulting to priority 5. You can set the thread priority with the setPriority() method
    - The Thread.sleep() method pauses a Thread for a specified period of time. For example, calling Thread.sleep(1000); pauses the thread for one second. Keep in mind that Thread.sleep() throws an InterruptedException, so be sure to surround it with a try/catch block
* Types of Exceptions
  + Checked – checked when compiled. Your code will not compile until you’ve handled the exception
    - Ex: InterruptedException
  + Unchecked – checked at runtime
    - Ex: ArithmeticException
* ArrayList
  + Unlike Array, ArrayLists can expand or shrink in size. ArrayLists store objects
    - Ex:
    - Allow duplicates, maintains the insertion order, and is non-synchronized
      * Non-synchronized – not thread safe
  + Methods:
    - contains() – returns true if the list contains the specified element
    - get(int index) – returns the element at the specified position in the list
    - size() – returns the number of elements in the list
    - clear() – removes all of the elements from the list
* LinkedList – very similar in syntax to the ArrayList
  + You cannot specify an initial capacity for the LinkedList
  + Stores the memory address (or link) of the element that follows it
  + 
  + Allow duplicate and is ordered
* ArrayList vs. LinkedList
  + ArrayList is better for storing and accessing data, as it is very similar to a normal array
  + LinkedList is better for manipulating data, such as making numerous inserts and deletes
* List interface
  + Classes: ArrayList, LinkedList, Vector, Stack
* HashMap – used for storing data collections as key and value pairs. One object is used as a key (index) to another object (the value)
  + Methods: put (add), remove (delete), get (access), containsKey/containsValue
  + 
  + Cannot contain duplicate keys
* Set – a collection that cannot contain duplicate elements. It models the mathematical set abstraction
  + One of the implementations of the Set is the HashSet class
  + 
    - The HashSet class does not automatically retain the order of the elements as they’re added. To order the elements, use a LinkedHashSet, which maintains a linked list of the set’s elements in the order in which they were inserted
* Sorting lists
  + Use the Collections class method, sort()
  + The methods in the Collections class are static, so you don’t need a Collections object to call them
  + Ex: Collections.sort(animals);
  + Other methods:
    - max(Collection c)
    - min (Collection c)
    - reverse(List list)
    - shuffle(List list)
* Collections – a framework that has a set of classes and interfaces that implement commonly reusable collection data structure
  + Store and manipulate a group of objects
* Iterator – an object that enables to cycle through a collection, obtain or remove elements
  + Before you can access a collection through an iterator, you must obtain one. Each of the collection classes provides an iterator() method that returns an iterator to the start of the collection. By using this iterator object, you can access each element in the collection, one element at a time
  + Methods:
    - hasNext() – returns true if there is at least one more element; otherwise, returns false
    - next() – returns the next object and advances the iterator
    - remove() – removes the last object that was returned by next from the collection
  + 
* Debugger
  + Evaluate expression – to evaluate an expression in the IntelliJ debugger:
    - 1. Start the application in debugger
    - 2. Navigate to Run -> Evaluate expression
    - 3. Enter expression
  + IntelliJ debugger – allow for stopping the execution of a program at certain points, inspect variables, and step into functions/methods to see what the program is doing
  + Breakpoint – allows for stopping the program execution at certain points
    - Can be set by hovering the mouse over the editor’s gutter area and clicking on it. The breakpoint will be identified by red circle symbols. To remove the break point you will just click on the red circle symbol
  + Step into action
    - While in debugger mode, if you have a breakpoint placed in front of a method, the application will stop when it gets to the breakpoint before the method. At that time, you can select the step into lab that is part of the debugger to step through each line of the method and you will continue stepping through each line of code for the rest of the program
  + Step out action
    - The same as the step into action. The key is that if you place the breakpoint in front of the method you are wanting to debug, the application will stop
  + Step over action
    - Does not enter into a function, instead, it will jump to the next line of code
  + Resume program action
    - Will continue the execution of a program by ignoring all the breakpoints
  + Smart step into action
    - While debugging, you may sometimes reach a line of code that calls several methods. When debugging these lines of code, the debugger typically allows you to use step into and leads you through all child functions and then back to the parent function. However, what if we only wanted to step into one child function? With Smart step-into, it allows you to choose the function to step into