Runners
  => CommandLineRunner(I)[1.0V]
  => ApplicationRunner(I)[1.3V]

The difference b/w the above 2 runners is they vary in the way the arguments are collected.

ApplicationArguments
====================
  1. Option Arguments(inputs)
            --key = val

  2. Non Option Arguments(commands)
            start
            close
          data
            execute
          dump
            stop

Difference b/w CLR and APR?
  Ans. Working process same, only difference is in inputs.
        CLR reads inputs from main() and stores as ArrayFormat(String[])
        ALR reads inputs from main() and converts into Option(--key=value) and Non-Option(only value) args and given to AR.

Note: ApplicationRunner can be used to override the value which are coming from application.properties file aslo.

CommandLineRunner
=================
```
@FunctionalInterface
public interface CommandLineRunner {
      void run(String... args) throws Exception;
}
++++++++++++++++++++++++++++++++++++++++++++++++++
SourceCode1::
@Configuration
public class AppConfig {
      @Bean
      public CommandLineRunner runA() {
            // syntax:: new CommandLineRunner(){....}
            return new CommandLineRunner() {
                  @Override
                  public void run(String... args) throws Exception {
                        System.out.println("HELLO :: " + Arrays.asList(args));
                  }
            };
      }
}

SoureCode::2
@Configuration
public class AppConfig {
      @Bean
      public CommandLineRunner runA() {
            // Syntax:: Interface obj =(params)->{MethodBody;}
```

```
		return (args) -> {
			System.out.println("FROM RUNNER :: " + Arrays.asList(args));
		};
	}
}

SourceCode:: 3
@Configuration
public class AppConfig {
	@Bean
	public CommandLineRunner runA() {
		//Syntax=> ClassName :: methodName
		return SampleTest::test;
	}
}
public class SampleTest {
	public static void test(String... args) {
		System.out.println("FROM METHOD REF:: " + Arrays.asList(args));
	}
}
```

refer:: SpringBoot-Runner-02,SpringBoot-Runner-03

Adding Custom banner to SpringBoot application

```
public ConfigurableApplicationContext run(String... args) {

		long startTime = System.nanoTime();
		DefaultBootstrapContext bootstrapContext = createBootstrapContext();
		ConfigurableApplicationContext context = null;
		configureHeadlessProperty();

		SpringApplicationRunListeners listeners = getRunListeners(args);
		listeners.starting(bootstrapContext, this.mainApplicationClass);

		try {
			ApplicationArguments applicationArguments = new
DefaultApplicationArguments(args);
			ConfigurableEnvironment environment =
prepareEnvironment(listeners, bootstrapContext, applicationArguments);
			configureIgnoreBeanInfo(environment);

			//Banner is getting generated
			Banner printedBanner = printBanner(environment);

			context = createApplicationContext();
			context.setApplicationStartup(this.applicationStartup);
			prepareContext(bootstrapContext, context, environment, listeners,
applicationArguments, printedBanner);
			refreshContext(context);
			afterRefresh(context, applicationArguments);
			Duration timeTakenToStartup = Duration.ofNanos(System.nanoTime()
- startTime);
			if (this.logStartupInfo) {
				new
StartupInfoLogger(this.mainApplicationClass).logStarted(getApplicationLog(),
timeTakenToStartup);
			}
```

```
                  listeners.started(context, timeTakenToStartup);
                  callRunners(context, applicationArguments);
            }
            catch (Throwable ex) {
                  handleRunFailure(context, ex, listeners);
                  throw new IllegalStateException(ex);
            }
            try {
                  Duration timeTakenToReady = Duration.ofNanos(System.nanoTime() -
startTime);
                  listeners.ready(context, timeTakenToReady);
            }
            catch (Throwable ex) {
                  handleRunFailure(context, ex, null);
                  throw new IllegalStateException(ex);
            }
            return context;
      }
```

Creating our OwnBaner in Programmatic Approach

```
@FunctionalInterface
public interface Banner {

      /**
       * Print the banner to the specified print stream.
       * @param environment the spring environment
       * @param sourceClass the source class for the application
       * @param out the output print stream
       */
      void printBanner(Environment environment, Class<?> sourceClass, PrintStream
out);

}
```

Making the Banner to display while booting the Application

```
enum Mode {

            /**
             * Disable printing of the banner.
             */
            OFF,

            /**
             * Print the banner to System.out.
             */
            CONSOLE,

            /**
             * Print the banner to the log file.
             */
            LOG

      }
```

Step1::
  src/main/java

```
            |=> in.ineuron.banner
                              |=> banner.txt


banner.txt
===========
 __ __  __  ____ __ __ ____    ___   __  __
 || ||\ || ||    || || || \\  // \\  ||\ ||
 || ||\\|| ||==  || || ||_// ((   )) ||\\||
 || || \|| ||___ \\_// || \\  \\_//  || \||


Step2::
application.properties
======================
spring.main.banner-mode=console
spring.banner.location=classpath:in/ineuron/banner/banner.txt

Step3::
@SpringBootApplication
public class SpringBootBannerAppApplication {

      @SuppressWarnings("static-access")
      public static void main(String[] args) {
            SpringApplication application = new SpringApplication();
            ConfigurableApplicationContext ctx =
application.run(SpringBootBannerAppApplication.class, args);
            ctx.close();
      }
}

Step4:: Run the application and check the banner in console.


Working with SpringBoot-Mail
============================
1. Add the following dependancies

<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-mail</artifactId>
</dependency>

2. Required Bean JavaMailSender will be autoconfigured

3. Create a Service class

public interface IPurchaseOrder {
      public String purchase(String[] items,double [] prices,String[] toEmails)
throws Exception;

}

@Service("service")
public class PurchaseOrderImpl implements IPurchaseOrder {

      @Autowired
      private JavaMailSender sender;
```

```
	@Value("${spring.mail.username}")
	private String fromEmail;

	@Override
	public String purchase(String[] items, double[] prices, String[] toEmails)
throws Exception {

		double amt = 0.0;
		for (double price : prices) {
			amt = amt + price;
		}

		String msg = Arrays.toString(items) + "with price :: " +
Arrays.toString(prices)
				+ " are purchase with billamout :: " + amt;

		String status = sendMail(msg, toEmails);

		return msg + "---> " + status;
	}

	private String sendMail(String msg, String[] toEmails) throws Exception {
		System.out.println("Sender implementation class is :: " +
sender.getClass().getName());

		MimeMessage message = sender.createMimeMessage();
		MimeMessageHelper helper = new MimeMessageHelper(message, true);

		helper.setFrom(fromEmail);
		helper.setCc(toEmails);
		helper.setSubject("Open it to konw it");
		helper.setSentDate(new Date());
		helper.setText(msg);
		helper.addAttachment("ineuron.jpg", new
ClassPathResource("ineuron.jpg"));
		sender.send(message);

		return "mail-sent";
	}
}
```

4. Set up application.properties

```
#Properties to tell the mail protocol vendor
spring.mail.host=smtp.gmail.com
spring.mail.port=587

#Actual username,password of sender
spring.mail.username=setusername
spring.mail.password=setpassword

#Property to trigger smtp
spring.mail.properties.mail.smtp.auth=false
spring.mail.properties.mail.smtp.starttls.enable=true
spring.mail.properties.mail.smtp.starttls.required = true
```

5. Build main app and run the application

```
@SpringBootApplication
public class SpringBootMailAppApplication {

	public static void main(String[] args) {
		ApplicationContext context = 
SpringApplication.run(SpringBootMailAppApplication.class, args);

		IPurchaseOrder order = context.getBean(IPurchaseOrder.class);

		try {
			String msg = order.purchase(new String[] { "Fossil-Chronography", 
"USPOLO-Tshirt", "LouisPhilippe-Shoes" },
					new double[] { 12000.0, 5000.0, 6000.0 }, new 
String[] { "attrayaghoshdas@gmail.com",
							"nithinvanga01@gmail.com", 
"nilesh7345kaka@gmail.com" });

			System.out.println(msg);

		} catch (Exception e) {
			e.printStackTrace();
		}

		((ConfigurableApplicationContext) context).close();
	}
}
```

```
Note: Before running app
	a. login to sender email account(like gmail login),go to account letter/image
	b. go to manage settings -> security -> change less secure apps access to ON
```

Evening :: All the remaining topics of DataJpA(H2,postgresql,workign with mulitple database,Association)