```
#!/bin/sh

#
# Copyright © 2015-2021 the original authors.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#      https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

##############################################################################
#
#   Gradle start up script for POSIX generated by Gradle.
#
#   Important for running:
#
#   (1) You need a POSIX-compliant shell to run this script. If your /bin/sh is
#       noncompliant, but you have some other compliant shell such as ksh or
#       bash, then to run this script, type that shell name before the whole
#       command line, like:
#
#           ksh Gradle
#
#       Busybox and similar reduced shells will NOT work, because this script
#       requires all of these POSIX shell features:
#         * functions;
#         * expansions «$var», «${var}», «${var:-default}», «${var+SET}»,
#           «${var#prefix}», «${var%suffix}», and «$( cmd )»;
#         * compound commands having a testable exit status, especially «case»;
#         * various built-in commands including «command», «set», and «ulimit».
#
#   Important for patching:
#
#   (2) This script targets any POSIX shell, so it avoids extensions provided
#       by Bash, Ksh, etc; in particular arrays are avoided.
#
#       The "traditional" practice of packing multiple parameters into a
#       space-separated string is a well documented source of bugs and security
#       problems, so this is (mostly) avoided, by progressively accumulating
#       options in "$@", and eventually passing that to Java.
#
#       Where the inherited environment variables (DEFAULT_JVM_OPTS, JAVA_OPTS,
#       and GRADLE_OPTS) rely on word-splitting, this is performed explicitly;
#       see the in-line comments for details.
#
#       There are tweaks for specific operating systems such as AIX, CygWin,
#       Darwin, MinGW, and NonStop.
#
#   (3) This script is generated from the Groovy template
#       https://github.com/gradle/gradle/blob/master/subprojects/plugins/src/main/resources/org/gradle/a
#       within the Gradle project.
#
#       You can find Gradle at https://github.com/gradle/gradle/.
#
```

```
##############################################################################

# Attempt to set APP_HOME

# Resolve links: $0 may be a link
app_path=$0

# Need this for daisy-chained symlinks.
while
    APP_HOME=${app_path%"${app_path##*/}"}  # leaves a trailing /; empty if no leading path
    [ -h "$app_path" ]
do
    ls=$( ls -ld "$app_path" )
    link=${ls#*' -> '}
    case $link in             #(
      /*)   app_path=$link ;; #(
      *)    app_path=$APP_HOME$link ;;
    esac
done

APP_HOME=$( cd "${APP_HOME:-./}" && pwd -P ) || exit

APP_NAME="Gradle"
APP_BASE_NAME=${0##*/}

# Add default JVM options here. You can also use JAVA_OPTS and GRADLE_OPTS to pass JVM options to this s
DEFAULT_JVM_OPTS='"-Xmx64m" "-Xms64m"'

# Use the maximum available, or set MAX_FD != -1 to use that value.
MAX_FD=maximum

warn () {
    echo "$*"
} >&2

die () {
    echo
    echo "$*"
    echo
    exit 1
} >&2

# OS specific support (must be 'true' or 'false').
cygwin=false
msys=false
darwin=false
nonstop=false
case "$( uname )" in                #(
  CYGWIN* )         cygwin=true  ;; #(
  Darwin* )         darwin=true  ;; #(
  MSYS* | MINGW* )  msys=true    ;; #(
  NONSTOP* )        nonstop=true ;;
esac

CLASSPATH=$APP_HOME/gradle/wrapper/gradle-wrapper.jar


# Determine the Java command to use to start the JVM.
if [ -n "$JAVA_HOME" ] ; then
    if [ -x "$JAVA_HOME/jre/sh/java" ] ; then
        # IBM's JDK on AIX uses strange locations for the executables
        JAVACMD=$JAVA_HOME/jre/sh/java
```

```
    else
        JAVACMD=$JAVA_HOME/bin/java
    fi
    if [ ! -x "$JAVACMD" ] ; then
        die "ERROR: JAVA_HOME is set to an invalid directory: $JAVA_HOME

Please set the JAVA_HOME variable in your environment to match the
location of your Java installation."
    fi
else
    JAVACMD=java
    which java >/dev/null 2>&1 || die "ERROR: JAVA_HOME is not set and no 'java' command could be found

Please set the JAVA_HOME variable in your environment to match the
location of your Java installation."
fi

# Increase the maximum file descriptors if we can.
if ! "$cygwin" && ! "$darwin" && ! "$nonstop" ; then
    case $MAX_FD in #(
      max*)
        MAX_FD=$( ulimit -H -n ) ||
            warn "Could not query maximum file descriptor limit"
    esac
    case $MAX_FD in  #(
      '' | soft) :;; #(
      *)
        ulimit -n "$MAX_FD" ||
            warn "Could not set maximum file descriptor limit to $MAX_FD"
    esac
fi

# Collect all arguments for the java command, stacking in reverse order:
#   * args from the command line
#   * the main class name
#   * -classpath
#   * -D...appname settings
#   * --module-path (only if needed)
#   * DEFAULT_JVM_OPTS, JAVA_OPTS, and GRADLE_OPTS environment variables.

# For Cygwin or MSYS, switch paths to Windows format before running java
if "$cygwin" || "$msys" ; then
    APP_HOME=$( cygpath --path --mixed "$APP_HOME" )
    CLASSPATH=$( cygpath --path --mixed "$CLASSPATH" )

    JAVACMD=$( cygpath --unix "$JAVACMD" )

    # Now convert the arguments - kludge to limit ourselves to /bin/sh
    for arg do
        if
            case $arg in                                #(
              -*)   false ;;                            # don't mess with options #(
              /?*)  t=${arg#/} t=/${t%%/*}              # looks like a POSIX filepath
                    [ -e "$t" ] ;;                      #(
              *)    false ;;
            esac
        then
            arg=$( cygpath --path --ignore --mixed "$arg" )
        fi
        # Roll the args list around exactly as many times as the number of
        # args, so each arg winds up back in the position where it started, but
        # possibly modified.
```

```
        #
        # NB: a `for` loop captures its iteration list before it begins, so
        # changing the positional parameters here affects neither the number of
        # iterations, nor the values presented in `arg`.
        shift                   # remove old arg
        set -- "$@" "$arg"      # push replacement arg
    done
fi

# Collect all arguments for the java command;
#   * $DEFAULT_JVM_OPTS, $JAVA_OPTS, and $GRADLE_OPTS can contain fragments of
#     shell script including quotes and variable substitutions, so put them in
#     double quotes to make sure that they get re-expanded; and
#   * put everything else in single quotes, so that it's not re-expanded.

set -- \
        "-Dorg.gradle.appname=$APP_BASE_NAME" \
        -classpath "$CLASSPATH" \
        org.gradle.wrapper.GradleWrapperMain \
        "$@"

# Use "xargs" to parse quoted args.
#
# With -n1 it outputs one arg per line, with the quotes and backslashes removed.
#
# In Bash we could simply go:
#
#   readarray ARGS < <( xargs -n1 <<<"$var" ) &&
#   set -- "${ARGS[@]}" "$@"
#
# but POSIX shell has neither arrays nor command substitution, so instead we
# post-process each arg (as a line of input to sed) to backslash-escape any
# character that might be a shell metacharacter, then use eval to reverse
# that process (while maintaining the separation between arguments), and wrap
# the whole thing up as a single "set" statement.
#
# This will of course break if any of these variables contains a newline or
# an unmatched quote.
#

eval "set -- $(
        printf '%s\n' "$DEFAULT_JVM_OPTS $JAVA_OPTS $GRADLE_OPTS" |
        xargs -n1 |
        sed ' s~[^-[:alnum:]+,./:=@_]~\\&~g; ' |
        tr '\n' ' '
    )" '"$@"'

exec "$JAVACMD" "$@"
```

```
version=0.0.1
```

```
import java.util.Properties

/*
 * This file was generated by the Gradle 'init' task.
 *
 * This generated file contains a sample Kotlin application project to get you started.
 * For more details take a look at the 'Building Java & JVM projects' chapter in the Gradle
 * User Manual available at https://docs.gradle.org/7.2/userguide/building_java_projects.html
 */

// Define variables for project configurations
val projectGroup = "dev.onelenyk"
val mainAppClassName = "$projectGroup.pdfproject.AppKt"

// Read properties file
val versionProperties =
    Properties().apply {
        load(file("version.properties").inputStream())
    }

val projectVersion = versionProperties["version"] as String
plugins {
    id("org.jetbrains.kotlin.jvm") version "1.8.22"
    id("com.github.johnrengelman.shadow") version "7.0.0" // Shadow plugin for creating a fat JAR
    id("org.jlleitschuh.gradle.ktlint") version "12.1.0"
    id("org.jetbrains.dokka") version "1.9.10"
    `maven-publish` // Required for publishing the library
    application // Apply the application plugin to add support for building a CLI application in Java.
}

group = projectGroup
version = projectVersion

repositories {
    mavenCentral()
    maven(url = "https://jitpack.io")
}

dependencies {

    // Align versions of all Kotlin components
    implementation(platform("org.jetbrains.kotlin:kotlin-bom"))

    // Use the Kotlin JDK 8 standard library.
    implementation("org.jetbrains.kotlin:kotlin-stdlib-jdk8")

    // This dependency is used by the application.
    implementation("com.google.guava:guava:30.1.1-jre")

    // Use the Kotlin test library.
    testImplementation("org.jetbrains.kotlin:kotlin-test")

    // Use the Kotlin JUnit integration.
    testImplementation("org.jetbrains.kotlin:kotlin-test-junit")

    // cli library
    implementation("info.picocli:picocli:4.6.2")

    implementation("dev.onelenyk:gitignore-parser:v0.1.6")

    implementation("com.itextpdf:itext7-core:7.1.16")
    implementation("com.itextpdf:html2pdf:3.0.3")
```

```
}

application {
    // Define the main class for the application.
    mainClass.set(mainAppClassName)
}

tasks.shadowJar {
    archiveClassifier.set("")
    manifest {
        attributes["Main-Class"] = mainAppClassName
    }
}

tasks.withType<Jar> {
    manifest {
        attributes["Main-Class"] = mainAppClassName
        attributes["Implementation-Title"] = project.name
        attributes["Implementation-Version"] = project.version
    }
    archiveFileName.set("pdfproject.jar")
}

// dokka

tasks.register<Jar>("dokkaHtmlJar") {
    dependsOn(tasks.dokkaHtml)
    from(tasks.dokkaHtml.flatMap { it.outputDirectory })
    archiveClassifier.set("javadoc")
}

tasks.register<Jar>("dokkaJavadocJar") {
    dependsOn(tasks.dokkaJavadoc)
    from(tasks.dokkaJavadoc.flatMap { it.outputDirectory })
    archiveClassifier.set("javadoc")
}

// Configure publishing
publishing {
    publications {
        create<MavenPublication>("mavenJava") {
            from(components["java"])
            artifact(tasks["dokkaJavadocJar"])
        }
    }
}
```

```
package dev.onelenyk.pdfproject

import picocli.CommandLine
import java.nio.file.Paths
import java.util.concurrent.Callable

@CommandLine.Command(
    name = "pdfproject",
    mixinStandardHelpOptions = true,
    version = ["pdfproject 1.0"],
    description = ["A command-line tool to convert any project to a PDF document."],
)
class App : Callable<Int> {
    @CommandLine.Option(names = ["-p", "--projectRoot"], description = ["Path to the project root"], req
    lateinit var projectRoot: String

    @CommandLine.Option(names = ["-r", "--rules"], description = ["Custom rules for filtering files"], s
    var customRules: List<String> = defaultRules

    override fun call(): Int {
        val projectRootPath = Paths.get(projectRoot).toAbsolutePath()
        val converter =
            ProjectToPdfExporter(
                projectRootPath,
                customRules = customRules,
            )
        converter.executeConversion()
        return 0
    }

    companion object {
        val defaultRules =
            listOf(
                ".*\\.idea(/|$)",
                ".*\\.git(/|$)",
                ".*\\.jar$",
                ".*\\.DS_Store",
                ".*\\.pdf$",
                ".*\\.bat$",
            )
    }
}

fun main(args: Array<String>) {
    val exitCode = CommandLine(App()).execute(*args)
    System.exit(exitCode)
}

fun mainTest(args: Array<String>) {
    val projectRootPath = Paths.get("").toAbsolutePath()
    val converter = ProjectToPdfExporter(projectRootPath)
    converter.executeConversion()
}
```

```
package dev.onelenyk.pdfproject

import com.itextpdf.html2pdf.HtmlConverter
import com.itextpdf.kernel.pdf.PdfDocument
import com.itextpdf.kernel.pdf.PdfReader
import com.itextpdf.kernel.pdf.PdfWriter
import com.itextpdf.kernel.utils.PdfMerger
import dev.onelenyk.FileProcessor
import dev.onelenyk.IFileProcessor
import java.io.File
import java.io.FileInputStream
import java.io.FileOutputStream
import java.nio.file.Path

class ProjectToPdfExporter(
    private val rootDirectory: Path,
    customRules: List<String> = listOf(),
) {
    private val fileProcessor: IFileProcessor =
        FileProcessor(
            rootDirectory,
            customRules = customRules,
        )

    private val outputFilePath: String = "$rootDirectory/output.pdf"
    private val temporaryDirectoryPath: String = "$rootDirectory/exporter"

    fun executeConversion() {
        println("Starting conversion...")
        convertProjectToHTML()
        compileHTMLToPDF()
        println("Conversion completed.")
    }

    private fun convertProjectToHTML() {
        setupTemporaryDirectory()

        val filesToProcess = fileProcessor.process()
        println("Converting files to HTML...")

        filesToProcess.forEachIndexed { index, file ->
            val relativePath = rootDirectory.parent.resolve(file)
            val htmlFilePath = "$temporaryDirectoryPath/$file.html"
            File(htmlFilePath).apply {
                parentFile.mkdirs()
                writeText(convertFileToHTML(relativePath.toFile()))
            }
            println("Converted [${index + 1}/${filesToProcess.size}] files to HTML.")
        }
    }

    private fun compileHTMLToPDF() {
        val htmlFiles = findHtmlFiles()
        println("Compiling HTML files into PDF... Total files: ${htmlFiles.size}")

        val tempPdfFiles = convertHtmlToPdf(htmlFiles)

        println("HTML to PDF conversion completed. Merging PDF files...")
        mergePdfFiles(tempPdfFiles)

        cleanUpTemporaryFiles(tempPdfFiles)
        println("Compiled HTML to PDF successfully. Output file: $outputFilePath")
```

```
    }

    private fun setupTemporaryDirectory() {
        File(temporaryDirectoryPath).apply {
            if (exists()) deleteRecursively()
            mkdirs()
        }
    }

    private fun findHtmlFiles(): List<File> =
        File(temporaryDirectoryPath)
            .listFiles()
            ?.flatMap { it.walk().filter { file -> file.extension == "html" } }
            .orEmpty()

    private fun convertHtmlToPdf(htmlFiles: List<File>): List<File> =
        htmlFiles.mapIndexed { index, htmlFile ->
            println("Converting HTML to PDF: ${index + 1}/${htmlFiles.size} (${htmlFile.name})")
            File(temporaryDirectoryPath, "temp_$index.pdf").apply {
                HtmlConverter.convertToPdf(FileInputStream(htmlFile), FileOutputStream(this))
                htmlFile.delete() // Consider deleting this line if you want to keep HTML files for revi
            }
        }

    private fun mergePdfFiles(tempPdfFiles: List<File>) {
        PdfDocument(PdfWriter(outputFilePath)).use { finalPdf ->
            val pdfMerger = PdfMerger(finalPdf)
            tempPdfFiles.forEachIndexed { index, tempPdfFile ->
                println("Merging PDF file ${index + 1} of ${tempPdfFiles.size}")
                PdfDocument(PdfReader(tempPdfFile.absolutePath)).use { tempPdfDocument ->
                    pdfMerger.merge(tempPdfDocument, 1, tempPdfDocument.numberOfPages)
                }
                tempPdfFile.delete() // Ensure temporary PDF files are cleaned up after merging.
            }
        }
        println("All PDF files have been merged into $outputFilePath")
    }

    private fun cleanUpTemporaryFiles(tempPdfFiles: List<File>) {
        tempPdfFiles.forEach(File::delete)
        File(temporaryDirectoryPath).deleteRecursively()
    }

    private fun convertFileToHTML(file: File): String {
        val fileContent = file.readText(Charsets.UTF_8)
        return "<html><body><pre><code>${escapeHtml(fileContent)}</code></pre></body></html>"
    }

    private fun escapeHtml(text: String): String =
        text.replace("&", "&amp;")
            .replace("<", "&lt;")
            .replace(">", "&gt;")
}
```

```
rootProject.name = "pdfproject"
include("app")
```

```
.gradle
build
!gradle/wrapper/gradle-wrapper.jar
!**/src/main/**/build/
!**/src/test/**/build/

### IntelliJ IDEA ###
.idea/modules.xml
.idea/jarRepositories.xml
.idea/compiler.xml
.idea/libraries/
*.iws
*.iml
*.ipr
out/
!**/src/main/**/out/
!**/src/test/**/out/

### Eclipse ###
.apt_generated
.classpath
.factorypath
.project
.settings
.springBeans
.sts4-cache
bin/
!**/src/main/**/bin/
!**/src/test/**/bin/

### NetBeans ###
/nbproject/private/
/nbbuild/
/dist/
/nbdist/
/.nb-gradle/

### VS Code ###
.vscode/

### Mac OS ###
.DS_Store
/.idea/
```

```
distributionBase=GRADLE_USER_HOME
distributionPath=wrapper/dists
distributionUrl=https\://services.gradle.org/distributions/gradle-8.2-bin.zip
zipStoreBase=GRADLE_USER_HOME
zipStorePath=wrapper/dists
```

```
#!/bin/bash

# Variables
HOMEE="/Users/lenyk/IdeaProjects"

PROJECT_DIR="$HOMEE/pdf-project"
JAR_PATH="$PROJECT_DIR/pdfproject.jar"
GIT_REPO_URL="https://github.com/onelenyk/pdf-project.git"
TARGET_SCRIPT="/usr/local/bin/pdfproject"

# Function to check if JAR file exists
check_jar_exists() {
    if [ -f "$JAR_PATH" ]; then
        echo "JAR file exists at $JAR_PATH."
        return 0
    else
        return 1
    fi
}

# Function to clone the repository and build the project
clone_and_build_project() {
    echo "Cloning repository from $GIT_REPO_URL..."
    git clone "$GIT_REPO_URL" "$PROJECT_DIR"

    echo "Building the project..."
    cd "$PROJECT_DIR" || exit
    ./gradlew clean build

    # Move the built JAR to the desired location
    mv "$PROJECT_DIR/app/build/libs/kotlin-cli-executor.jar" "$JAR_PATH"
}

# Function to create the execution script
create_execution_script() {
    echo "Creating execution script at $TARGET_SCRIPT..."

    # Content of the script
    SCRIPT_CONTENT="#!/bin/bash
    # Define the path to the JAR file
    JAR_PATH=\"${JAR_PATH}\"

    # Run the JAR file
    java -jar \"\$JAR_PATH\" \"\$@\"
    "

    # Create the script in /usr/local/bin
    echo "$SCRIPT_CONTENT" | sudo tee "$TARGET_SCRIPT" > /dev/null

    # Make the script executable
    sudo chmod +x "$TARGET_SCRIPT"

    echo "Script created at $TARGET_SCRIPT and made executable."
}

# Main script logic
if check_jar_exists; then
    echo "JAR file is already present. No need to download."
else
    echo "JAR file not found. Downloading and building the project..."
    clone_and_build_project
fi
```

```
# Create the execution script
create_execution_script
```

```
kotlin.code.style=official
```