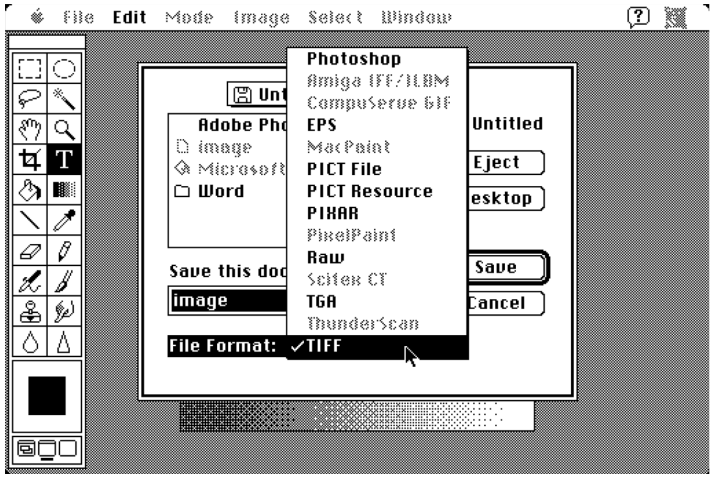
Building Desktop Applications

CS 346 Application Development

Desktop == “WIMP” interfaces

- Paradigm designed at Xerox PARC in 1973.
  - See also desktop metaphor in computing.
- Popularized with Apple Macintosh in 1984.
- **W**indows, **I**cons, **M**enus, **P**ointer
  - Each program runs in a self-contained and isolated **Window**.
  - **Icons** represent actions e.g., printer, trash can.
  - **Menus** represent commands that can be issued by the user.
  - **Pointer** refers to the mouse-pointer.
- Advantages: Discoverable, Simple, Familiar.
- Disadvantages: Resources, Accessibility.



Macintosh user interface from 1984.

Desktop-Specific Features

1. Graphical user interfaces (GUI)

- Windows as a logical abstraction; overlapping, min/max
- Reusable ”widgets” that we associate with desktop applications.

2. Keyboard + mouse interaction

- Keyboard shortcuts e.g., CMD-H to hide a window.
- Menus e.g., File, Edit, View, Window.
- Features: undo/redo, copy/paste, drag/drop.

All of these are provided by Compose Multiplatform.

Getting started

How to create a desktop project.

Step 1: Check dependencies

Update the **version catalog** in a standard Gradle project:

libs.versions.toml

```
[versions]
kotlin-ver = "2.0.20"
compose-plugin = "1.6.11"

[plugins]
jetbrains-compose = { id = "org.jetbrains.compose", version.ref = "compose-plugin" }
compose-compiler = { id = "org.jetbrains.kotlin.plugin.compose", version.ref = "kotlin-ver” }
```

Step 2: Add dependencies

Update the **build.gradle.kts** to include them:

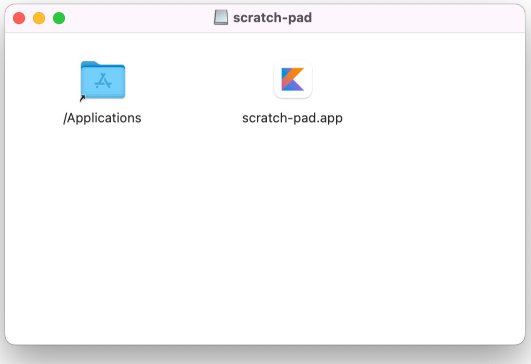
build.gradle.kts

```
plugins {
    alias(libs.plugins.kotlin.jvm)
    alias(libs.plugins.jetbrains.compose)
    alias(libs.plugins.compose.compiler)
}


dependencies {
    implementation (compose.desktop.currentOs)
}

compose.desktop {
    application {
        mainClass = “MainKt"
    }
}
```

Changes to Gradle?

Use the Gradle menu (`View > Tool Windows > Gradle`).

Command	What does it do?
`Tasks > build > clean`	Removes temp files (deletes the `/build` directory)
`Tasks > build > build`	Compiles your application
`Tasks > compose desktop > run`	Executes your application (builds it first if necessary)
`Tasks > compose desktop > package`	Create an installer for your platform!



Builds for your computer:

- MSI for Windows
- DMG for macOS
- DEB for Linux

Desktop Composables

Components specific to desktop applications.

Window

- A window is the top-level of our scene-graph.
  - We can have multiple windows.
  - Most applications require at least one window.
  - You programmatically manage windows and their contents.

- “Regular” window behavior is handled by the OS/toolkit.
  - e.g., resizing, dragging, minimize, maximize.

- Parameters
  - `Title` - Window title
  - `onCloseRequest` - lambda or function name to call on close
  - `State` - WindowState, including dimensions, position
  - *`contents`* - Window contents (also pass as trailing lambda)

Window

Independent, each has its own scene-graph.

```
fun main() {
    application {
        MaterialTheme {
            Window(
                title = "Window 1",
                onCloseRequest = ::exitApplication
            ) {
                Text("This is a window") // contents are a trailing lambda
            }

            Window(
                title = "Window 2",
                onCloseRequest = ::exitApplication
            ) {
                Text("This is also a window")
            }
        }
    }
}
```

samples/desktop/desktop-compose -> run **MultipleWindows** main method

(WindowState)

Controls position, size.

```
fun main() {
    application {
        MaterialTheme{
            Window(
                title = "WindowState",
                state = WindowState( // state obj manages size and position
                    position = WindowPosition(Alignment.center),
                    size = DpSize(300.dp, 200.dp)
                ),
                onCloseRequest = ::exitApplication
            ) {
                Text("This is a window")
            }
        }
    }
}
```

samples/desktop/desktop-compose -> run **WindowState** main method

Dialog Box

Foreground modal window

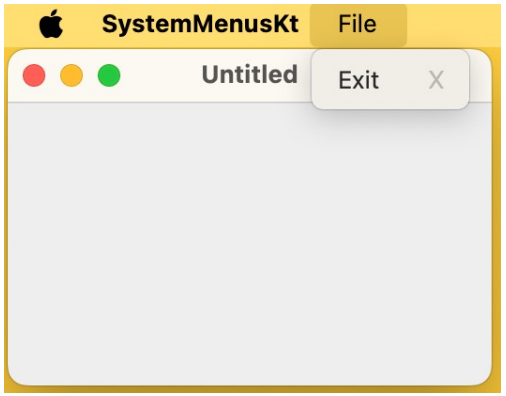
```
Window(
    title = "Main Window",
    onCloseRequest = ::exitApplication,
    state = WindowState(position = WindowPosition(Alignment.Center))
) {

    var isDialogOpen by remember { mutableStateOf(false) }
    Button(onClick = { isDialogOpen = true }) { // button sets a flag to show dialog
        Text(text = "Open dialog")
    }

    if (isDialogOpen) { // flag determines if this gets show
        DialogWindow(
            title = "Dialog Window",
            onCloseRequest = { isDialogOpen = false },
            state = rememberDialogState(position = WindowPosition(Alignment.Center))
        ) {
            Text("Dialog text goes here")
        }
    }
}
```

samples/desktop/desktop-compose -> run **Dialogs** main method

System Menu

OS determines position

```
fun main() = application {
    Window(onCloseRequest = ::exitApplication) {
        App(this@Window, this@application)
    }
}

@Composable
fun App(
    windowScope: FrameWindowScope,
    appScope: ApplicationScope
) {
    windowScope.MenuBar {
        Menu("File", mnemonic = 'F') {
            val nextWindowState = rememberWindowState()
            Item(
                "Exit",
                onClick = { appScope.exitApplication() },
                shortcut = KeyShortcut(
                                Key.X,
                                ctrl = false)
            )
        }
    }
}
```

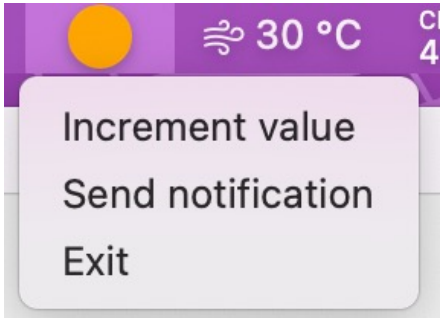


samples/desktop/desktop-compose -> run **SystemMenus** main method

System Tray

Taskbar or system tray icon

```
val trayState = rememberTrayState()
val notification = rememberNotification(
    "Notification", "Message from MyApp!"
)

Tray(
    state = trayState,
    icon = TrayIcon,
    menu = {
        Item("Increment value", onClick = { count++})
        Item("Send notification", onClick = {
            trayState.sendNotification(notification)
        })
        Item("Exit", onClick = { isOpen = false })
)
```



samples/desktop/desktop-compose -> run **SystemTray** main method

Interaction

Handling mouse and keyboard input on desktop.

Keyboard Input

```
fun main() = application {
    Window(
       title = "Key Events",
       state = WindowState(width = 500.dp, height = 100.dp),
       onCloseRequest = ::exitApplication,
       onKeyEvent = {
            if (it.type == KeyEventType.KeyUp) {
                println(it.key)
            }
        }
    ){
        val text = remember { mutableStateOf("") }
        val textField = TextField(
            value = text.value,
            onValueChange = { text.value = it }
        )
    }
}
```

Window-level event handler

Widget-level event handler

samples/desktop/desktop-compose -> run **Interaction** main method

Mouse Clicks

```
Box(
    modifier = Modifier
        .background(Color.Magenta)
        .fillMaxWidth(0.9f)
        .fillMaxHeight(0.2f)
        .combinedClickable(
            onClick = { text = "Click! ${count++}" },
            onDoubleClick = { text = "Double click! ${count++}" },
            onLongClick = { text = "Long click! ${count++}" }
        )
)
```

Multi-event handler necessary to handle all mouse inputs.

samples/desktop/desktop-compose -> run **Interaction** main method

Mouse Movement

```
var color by remember { mutableStateOf(Color(0, 0, 0)) }

Box(
    modifier = Modifier
        .background(Color.Magenta)
        .fillMaxWidth(0.9f)
        .fillMaxHeight(0.2f)
        .onPointerEvent(PointerEventType.Move) {
            val position = it.changes.first().position
            color = Color(
                position.x.toInt() % 256,
                position.y.toInt() % 256, 0
            )
        }
)
```

Drag handler. `it` contains a list of mouse movements.

lectures/desktop -> run **Interaction** main methods

Reference

- Bolt UIX. 2025. KMP: What You Can Only Do in desktopMain
- JetBrains. 2025. Compose Multiplatform Documentation.
- JetBrains. 2025. Kotlin Multiplatform Documentation.