Samsung R&D Institute Poland

Java Tech Guide

Poszerzaj i utrwalaj znajomość zagadnień technicznych, korzystając z praktycznych wskazówek naszych ekspertów.

Samsung Tech Guides

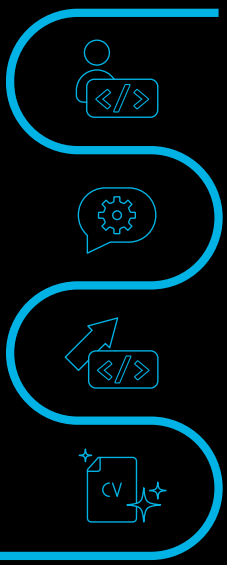
Drogi kandydacie!
Droga kandydatko!

Zapraszamy Cię do zapoznania się z Java Technical Guide - technicznym przewodnikiem poświęconym jednemu z najbardziej popularnych języków programowania!

Znajdziesz w nim tematy, które warto powtórzyć przed spotkaniem technicznym w rekrutacji do Samsung R&D!

Dodatkowo nasi Inżynierowie przygotowali przykładowe zadania testowe, które pomogą Ci sprawdzić swoją wiedzę.

Java Technical Guide - Dla kogo?



dla kandydatów aplikujących na stanowiska, w których wymagana jest znajomość Javy,

dla kandydatów zaproszonych już na spotkanie techniczne do Samsung R&D,

dla początkujących Inżynierów chcących poszerzyć/utrwalić swoją wiedzę z Javy,

dla osób zainteresowanych w przyszłości rekrutacją do Samsung R&D.

Powodzenia!

Java - podstawy

Praktyczne wykorzystanie języka wymaga dobrego opanowania jego podstawowych założeń. Konieczne jest uprzednie zapoznanie się z następującymi zagadnieniami:

- zmienne i ich zasięg,
- typy prymitywne,
- klasa String i operacje na niej,
- typy Enum,
- instrukcje sterujące i pętle.

Hierarchia klas

W języku obiektowym, jakim jest Java, kluczową rolę odgrywają klasy. Niezbędne w pracy jest, aby wiedzieć, jak działa w języku Java ich hierarchia, w szczególności:

- tworzenie typów pochodnych,
- różnice między interfejsem a klasą abstrakcyjną,
- jak działa rzutowanie typów,
- na czym polega polimorfizm,
- czym są anonimowe klasy wewnętrzne,
- czym są typy szablonowe (ang. Generics).

Struktury danych

Język Java udostępnia implementacje powszechnie stosowanych struktur danych. Od kandydatów oczekuje się, by znać przeznaczenie i sposób ich użycia, w szczególności:

- Jakie struktury danych zaimplementowane są w bibliotece standardowej Javy?
- Co jest interfejsem, a co konkretną implementacją?
- W jakich sytuacjach stosuje się poszczególne struktury, jakie są ich mocne i słabe strony (np. złożoność czasowa przy konkretnych operacjach)?

Przetwarzanie współbieżne

Niejednokrotnie do czynienia mamy z sytuacjami, w których tworzony program nie może lub nie powinien wykonywać operacji w sposób sekwencyjny. Konieczne jest wykonywanie wielu zadań równocześnie. Przykładem niech będą sytuacje, w których aplikacja musi reagować na zdarzenia interfejsu użytkownika. Kandydaci powinni wykazać się znajomością mechanizmów umożliwiających wdrożenie takich rozwiązań:

- pojęcie wątków i związaną z nim klasę Thread oraz podstawowe operacje na niej (takie jak sleep czy wait),
- mechanizmy synchronizacji, w tym słowo kluczowe „synchronized" oraz pojęcie monitora,
- sposoby umożliwiające zebranie wyników przetwarzania z wielu wątków, np. Future.

System kontroli wersji (Git™)

Aby efektywnie zarządzać bazą kodu i synchronizować pracę wielu osób, wykorzystuje się istniejące systemy kontroli wersji. Wykorzystywanym narzędziem jest przede wszystkim Git, tak więc kluczowe jest, aby znać:

podstawowe założenia (takie jak branch, czym różni się merge od rebase),

komendy (np. checkout, add, commit) w stopniu umożliwiającym sprawną pracę,

sposób pisania treściwej, lecz zarazem znaczącej treści commitu,

narzędzie bazujące na systemie kontroli wersji.

podstawowy „flow" pracy w projekcie z głównym branchem main (lub master), do którego składane są Pull Requesty (PR) bądź Code Review (CR),

Obiektowość

Obiektowość to nie tylko metoda strukturyzacji kodu, lecz też sposób formowania zależności pomiędzy danymi, przepływu informacji oraz dynamicznego wstrzykiwania implementacji. Pomocne do tego będzie zaznajomienie się z:

- filozofią obiektowości oraz sposobem rozwiązywania problemów i zależności pomiędzy obiektami,
- akronimem SOLID w celu wyjaśnienia i zastosowania poszczególnych jego składowych,
- różnicami pomiędzy klasą statyczną a klasą POJO,
- pojęciem - „klasa z danymi" (data class lub DTO), oraz opcjonalnie ze słowem kluczowym record od JDK 14,
- konsekwencjami wstrzykiwania obiektów przez dziedziczenie bądź kompozycje,
- rolą interfejsów.

Elementy programowania funkcyjnego i Stream API

Od wersji 8 do języka zostały wprowadzone pewne elementy programowania funkcyjnego, umożliwiające inne podejście do wykonywania operacji. Częścią tej zmiany jest również Stream API (nie mylić z klasami wejścia/wyjścia istniejącymi wcześniej), umożliwiające wykonywanie operacji na kolekcjach. Kandydujący powinni uprzednio poznać następujące powiązane zagadnienia:

- wyrażenia lambda,
- interfejsy funkcjonalne,
- zasady działania obiektów Stream,
- podstawowe operacje na obiektach Stream takie jak map, flatmap, filter.

Operacje na plikach

Większość programów nie obędzie się bez wczytywania bądź zapisywania plików lub strumieni danych z zewnątrz programu. Przydatne do tego będzie:

- biblioteka standardowa NIO oraz klasyczne Input/OutputStreamy,
- znajomość formatu JSON oraz sposobu przełożenia go na obiekty Java,
- użycie folderu resources w strukturze projektu.

Środowisko programi-styczne

W codziennej pracy wykorzystywane są dziś powszechnie środowiska programistyczne, ułatwiające w znacznym stopniu pisanie i budowanie kodu. W toku pracy niezbędne będzie, aby znać:

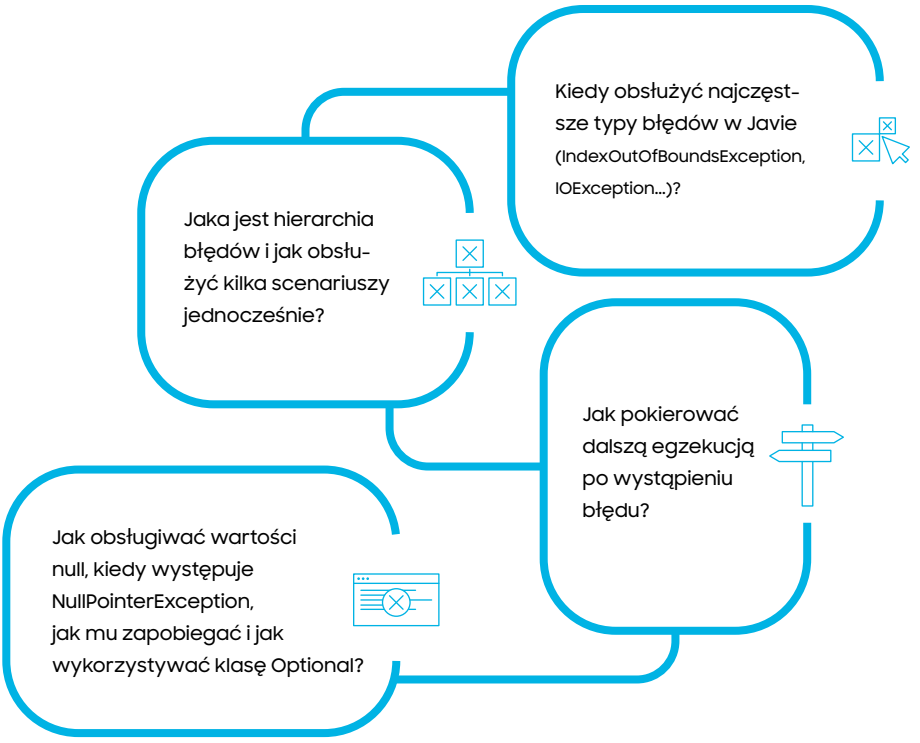
- środowiska IntelliJ IDEA oraz Android Studio [Android jest znakiem towarowym Google LLC],
- integrację z narzędziami budowania takimi jak Maven i Gradle,
- sposób uruchomienia swojego programu z konsoli (zarówno Linux jak i Windows).

Typowy dzień pracy programisty w znacznej mierze wypełnia również komunikacja ze Współpracownikami. Z tego względu warto poznać narzędzia wspomagające zarządzanie zadaniami, bądź wymianę informacji, zarówno z członkami zespołu, jak i z przełożonym:

- Jira
- Slack

Obsługa błędów

Integralną częścią logiki programu jest obsługa sytuacji niepożądanych, które mogą pojawić się w trakcie jego wykonywania. Przydatnym jest wiedzieć:



Analiza statyczna

Aby zredukować liczbę defektów w kodzie, stosuje się techniki automatycznej analizy. Pozwala ona ujawnić błędy, których istnienia developerzy często nie są świadomi. Od kandydatów oczekuję się, że będą wiedzieć:

- Czym jest analiza statyczna kodu, na co pozwala i jakie są jej ograniczenia?
- Jak różni się ona od analizy dynamicznej?
- Czym są narzędzia SonarQube bądź SonarLint i do czego są używane
- opcjonalnie, jak z powyższych narzędzi korzystać?

Dobre praktyki

Miarą jakości kodu jest nie tylko jego działanie, ale także łatwość w jego utrzymaniu, modyfikacji oraz szybkość zrozumienia przez innych. W tym celu konieczna jest czytelność i możliwie proste wprowadzanie zmian. W związku z tym kandydaci powinni posiadać rozeznanie w kwestii inżynierii kodu, w szczególności:

- ogólne praktyki dotyczące czystości kodu, takie jak dobór długości metod,
- nazewnictwo zmiennych,
- rozumieć, dlaczego komentarze w kodzie należy stosować oszczędnie i ostrożnie,
- wzorce projektowe np. Builder lub Singleton.

Testy jednostkowe

Testowanie jest często traktowane jako rzecz drugorzędna. Tymczasem testy pełnią ważną funkcję w cyklu życia projektu, z tego względu kandydaci powinni znać:

- znaczenie unit testów dla utrzymania projektu oraz pojęcie regresji,
- podstawy użycia biblioteki JUnit, takie jak metody assert i adnotacje, np. @Test, @BeforeEach,
- ogólne założenia metodyki TDD.

1.
Dane są tablice Stringów:

```
String authors[] = new String[] {„Parker, „Parker”,
„King”, „Homer”, „Hemingway”, „Goethe” , „Homer”};

String books[] = new String[] {„Devices and Desires”,
„Evil for Evil „, „The Dark Tower „, „Odyssey”,
„For whom the Bell Tolls”, „Faust”, „Iliad” };
```

Indeks nazwiska autora w tablicy authors, odpowiada indeksowi książki, którą napisał w tablicy books. Wynik działania programu powinien prezentować nazwisko danego autora tylko raz oraz tylko jeden z pojawiających się jego utworów.

2.
Dana jest mapa państw na ich stolice:

```
Map<String,String> countryToCapital = new HashMap<>();
countryToCapital.put („Poland”, „Warsaw”);
countryToCapital.put („France”, „Paris”);
countryToCapital.put („Germany”, „Berlin”);
countryToCapital.put („Spain”, „Madrid”);
countryToCapital.put („Czech Republic”, „Prague”);
countryToCapital.put („Italy”, „Rome”);
```

- Należy napisać kod, który zwróci listę tych krajów, których stolica zaczyna się na literę „P”.
- Na ile innych sposobów można wykonać to zadanie?
- Dlaczego są gorsze od zaproponowanego?

3.
Dany jest następujący fragment JSON:

```
company:”Samsung”,
    employee_count:3,
    “employees”:[
        {“name”:”Helen”, “email”:”helen@mail.com”},
        {“name”:”Bob”, “email”:”bob@mail.com”},
        {“name”:”Hailey”, “email”:”hailey@mail.com”}
    ]
```

- Należy stworzyć klasy w Javie, do których można byłoby zdeserializować ten fragment JSON.
- Jakich narzędzi i technik można użyć do samego procesu serializacji/deserializacji?
- Następnie do obiektu stworzonego na bazie powyższej klasy, należy dodać nowego pracownika na pozycję 3, zachowując pozostałych pracowników.
- Analogicznie, usunąć obiekt reprezentujący pracownika Bob.
- Czy da się uogólnić proces dodawania i usuwania do postaci funkcji, która działa na przekazanej w argumencie pozycji w liście?

4.
Dana jest klasa:

```
public class Untested {
    private int x;
    public Untested(int x){ this.x = x;}
    public Untested adjustX(int newX) {
        this.x = newX;
        return this;
    }
    public int getX(){ return x;}
    public boolean equals(Object obj){
        return this == obj
        ||obj instanceof Untested &&
((Untested)obj).x == this.x;
    }
}
```

Zadanie polega na napisaniu testu jednostkowego, który sprawdzi, czy wartość x dla instancji Untested tworzonej z argumentem 5, po wywołaniu metody adjustX() z argumentem 8, będzie rzeczywiście różna od początkowej.

Dodatkowo można sprawdzić, czy adjustX() na pewno zwraca ten sam obiekt co ten, na którym została wywołana.

5.
Dane są imiona klientów:

```
String [] customers = {“Harmony”, “Helga”, “Hermione”,
“Helen”, “Hailey”};
```

Zadanie polega na napisaniu kodu, który umożliwia przechowywanie struktury:

- zachowującej kolejność alfabetyczną imion,
- umożliwiającej łatwe (mało kosztowne algorytmicznie), usuwanie elementów z jakiejkolwiek pozycji.

6. Dany jest ciąg znaków:

```
String sentence = “We will steer SRPOL together to our future success.”
```

Jak można zaimplementować rozwiązanie, które:

- Zwraca sumaryczną długość 2 ostatnich słów?
- Zwraca sumaryczną liczbę wystąpień znaków „r” oraz „R”?

7.

Dana jest metoda obtainStringOrNull(), która z równym prawdopodobieństwem zwraca String lub null.

Nie używając instrukcji „if” należy zamienić zwrócony String na same wielkie litery lub jeżeli zwrócony String okaże się nullem, zastąpić go przez „nothing”.

8.

Rozwiązaniem tego zadania jest kod, który wyświetli asynchronicznie kolejne liczby od 1 do 10.

- Następnie wykona działanie sumowania w każdym wątku i zwróci wynik.
- Na koniec zsumuje wszystkie wyniki.

9.
Dana jest klasa StringProvider, a wewnątrz niej metoda z następującą sygnaturą:

```
private String fetchAvailableString() throws IOException
```

Klasa jest wykorzystana w następujący sposób:

```
public String getResourceString(ExternalResourcesManager
manager){
StringProvider provider = manager.accessStringProvider();
String availableString = provider.fetchAvailableString();
return availableString;
}
```

- Jak obsłużyć wyjątki, żeby program nie został zamknięty?
- Jak obsłużyć wyjątki w taki sposób, żeby zwrócić jak najbardziej precyzyjną informację o napotkanym problemie?

10.
Dane są następujące definicje:

```
interface X{}
interface Y extends X{}
interface Z{}

private static class A{
    A(){System.out.print(„A”);}
}
private static class B extends A implements Y {
    B(){System.out.print(„B”);}
}
private static class C implements Y {
    C(){printName();}

    protected void printName(){
    System.out.print(„C”);
    }}

private static class D extends B implements X,Z {}
private static class E extends C {
    protected void printName(){
        System.out.print(„E”);
    }
}
private static class F extends A implements Z {}
```

```
public static void main(String[] args){
    A a = _;
    C c = _;
    Y y = _;
    Z z = _;
    z = (Z) _;
    System.out.print(z instanceof Y);
}
```

Zadanie polega na uzupełnieniu luk w poniższym kodzie, tak aby po uruchomieniu programu otrzymać wynik „ABECAtrue”. Można stworzyć najwyżej cztery nowe obiekty.

11.

Należy napisać kod, który zwróci odpowiedź na pytanie:
Czy zawartość stworzonych obiektów typu Bug jest równa?

```
public class Main
    {
    public static void main(String[] args) {}

        Device deviceLocal = new Device(„Warsaw”);
        Device deviceRemote = new Device („Berlin”);

        Bug bugA = new
Bug(„OutOfMemoryError”,deviceLocal);
        Bug bugB = new
Bug(„OutOfMemoryError”,deviceRemote);
        Bug bugC = new
Bug(„NullPointerException”,deviceLocal);
    }
    public record Device(String name) {}
    public record Bug(String type, Device device) {}
```

- Jak powinno się zmienić tę funkcję, żeby nie uwzględniać wartości obiektu device w porównaniu?
- Jak należy zmodyfikować funkcję zwracającą równość, jeśli sygnatura klasy będzie jak poniżej:

```
public record Bug(String type, List<Device> device){}
```

Rekomendowane materiały online i książki

Git Immersion

The Linux command line

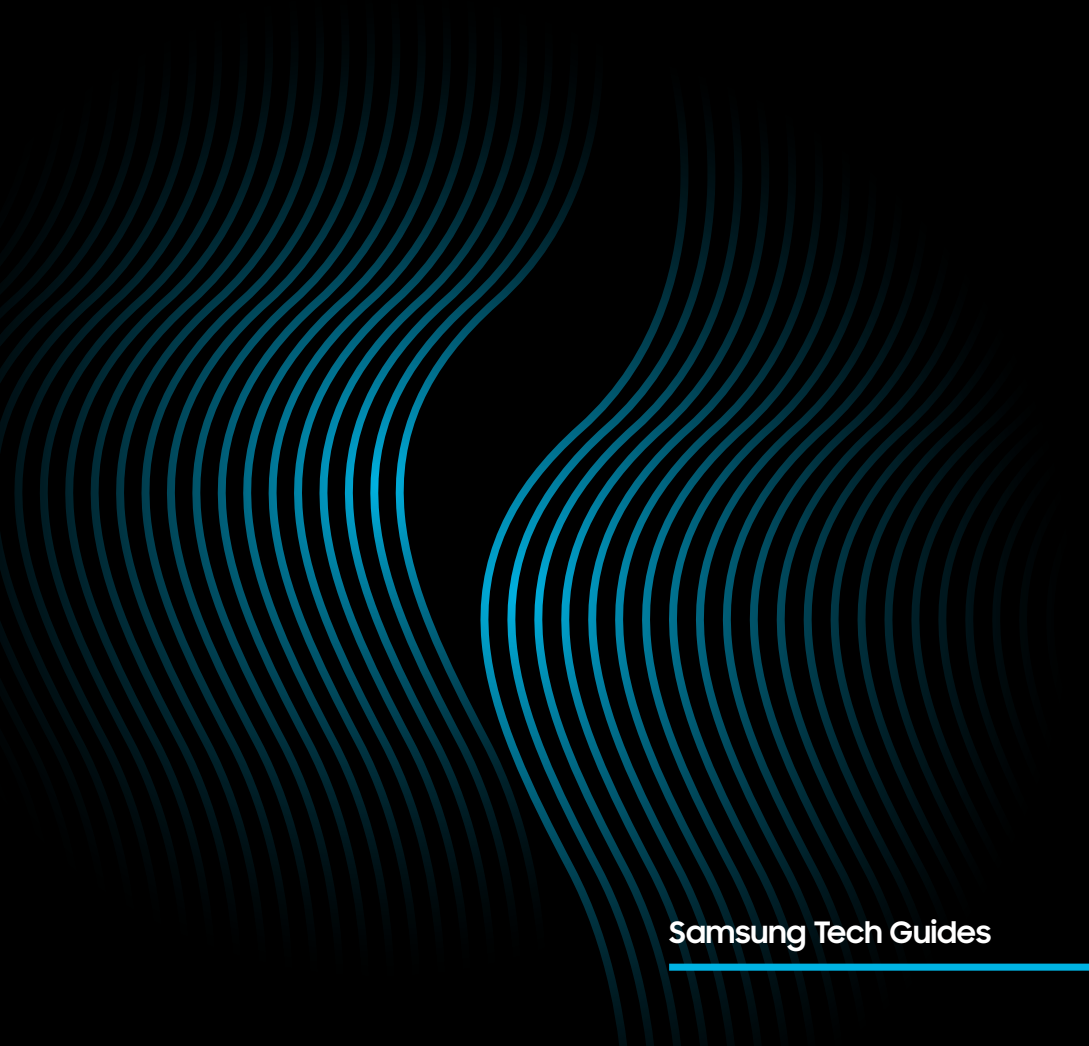
Jira Fundamentals

Scrum Guide

Robert C. Martin
„Clean Code" (Czysty kod)

Joshua Bloch
„Effective Java"
(Java. Efektywne programowanie)

Samsung R&D Institute Poland



Samsung Tech Guides