Problem 1600: Throne Inheritance

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

A kingdom consists of a king, his children, his grandchildren, and so on. Every once in a while, someone in the family dies or a child is born.

The kingdom has a well-defined order of inheritance that consists of the king as the first member. Let's define the recursive function

Successor(x, curOrder)

, which given a person

x

and the inheritance order so far, returns who should be the next person after

x

in the order of inheritance.

Successor(x, curOrder): if x has no children or all of x's children are in curOrder: if x is the king return null else return Successor(x's parent, curOrder) else return x's oldest child who's not in curOrder

For example, assume we have a kingdom that consists of the king, his children Alice and Bob (Alice is older than Bob), and finally Alice's son Jack.

In the beginning,

curOrder

will be

["king"]

.

Calling

Successor(king, curOrder)

will return Alice, so we append to

curOrder

to get

["king", "Alice"]

.

Calling

Successor(Alice, curOrder)

will return Jack, so we append to

curOrder

to get

["king", "Alice", "Jack"]

.

Calling

Successor(Jack, curOrder)

will return Bob, so we append to

curOrder

to get

["king", "Alice", "Jack", "Bob"]

.

Calling

Successor(Bob, curOrder)

will return

null

. Thus the order of inheritance will be

["king", "Alice", "Jack", "Bob"]

.

Using the above function, we can always obtain a unique order of inheritance.

Implement the

ThroneInheritance

class:

ThroneInheritance(string kingName)

Initializes an object of the

ThroneInheritance

class. The name of the king is given as part of the constructor.

void birth(string parentName, string childName)

Indicates that

parentName

gave birth to

childName

.

void death(string name)

Indicates the death of

name

. The death of the person doesn't affect the

Successor

function nor the current inheritance order. You can treat it as just marking the person as dead.

string[] getInheritanceOrder()

Returns a list representing the current order of inheritance

excluding

dead people.

Example 1:

Input

["ThroneInheritance", "birth", "birth", "birth", "birth", "birth", "birth", "getInheritanceOrder", "death", "getInheritanceOrder"] [["king"], ["king", "andy"], ["king", "bob"], ["king", "catherine"], ["andy", "matthew"], ["bob", "alex"], ["bob", "asha"], [null], ["bob"], [null]]

Output

[null, null, null, null, null, null, null, ["king", "andy", "matthew", "bob", "alex", "asha", "catherine"], null, ["king", "andy", "matthew", "alex", "asha", "catherine"]]

Explanation

ThroneInheritance t= new ThroneInheritance("king"); // order:

king

t.birth("king", "andy"); // order: king >

andy

t.birth("king", "bob"); // order: king > andy >

bob

t.birth("king", "catherine"); // order: king > andy > bob >

catherine

t.birth("andy", "matthew"); // order: king > andy >

matthew

> bob > catherine t.birth("bob", "alex"); // order: king > andy > matthew > bob >

alex

> catherine t.birth("bob", "asha"); // order: king > andy > matthew > bob > alex >

asha

> catherine t.getInheritanceOrder(); // return ["king", "andy", "matthew", "bob", "alex", "asha", "catherine"] t.death("bob"); // order: king > andy > matthew >

bob

> alex > asha > catherine t.getInheritanceOrder(); // return ["king", "andy", "matthew", "alex", "asha", "catherine"]

Constraints:

1 <= kingName.length, parentName.length, childName.length, name.length <= 15

kingName

,

parentName

,

childName

, and

name

consist of lowercase English letters only.

All arguments

childName

and

kingName

are

distinct

.

All

name

arguments of

death

will be passed to either the constructor or as

childName

to

birth

first.

For each call to

birth(parentName, childName)

, it is guaranteed that

parentName

is alive.

At most

10

5

calls will be made to

birth

and

death

.

At most

10

calls will be made to

getInheritanceOrder

.

Code Snippets

C++:

```
class ThroneInheritance {
public:
ThroneInheritance(string kingName) {

}

void birth(string parentName, string childName) {

}

void death(string name) {

}

vector<string> getInheritanceOrder() {

}
};
```

```
/**
* Your ThroneInheritance object will be instantiated and called as such:
* ThroneInheritance* obj = new ThroneInheritance(kingName);
* obj->birth(parentName,childName);
* obj->death(name);
* vector<string> param_3 = obj->getInheritanceOrder();
*/
```

Java:

```
class ThroneInheritance {

public ThroneInheritance(String kingName) {

}

public void birth(String parentName, String childName) {

}

public void death(String name) {

}

public List<String> getInheritanceOrder() {

}
}

/**
* Your ThroneInheritance object will be instantiated and called as such:
* ThroneInheritance obj = new ThroneInheritance(kingName);
* obj.birth(parentName,childName);
* obj.death(name);
* List<String> param_3 = obj.getInheritanceOrder();
*/
```

Python3:

```
class ThroneInheritance:
```

```
def __init__(self, kingName: str):


def birth(self, parentName: str, childName: str) -> None:


def death(self, name: str) -> None:


def getInheritanceOrder(self) -> List[str]:



# Your ThroneInheritance object will be instantiated and called as such:
# obj = ThroneInheritance(kingName)
# obj.birth(parentName,childName)
# obj.death(name)
# param_3 = obj.getInheritanceOrder()
```

Python:

```
class ThroneInheritance(object):

def __init__(self, kingName):
"""
:type kingName: str
"""


def birth(self, parentName, childName):
"""
:type parentName: str
:type childName: str
:rtype: None
"""


def death(self, name):
"""
:type name: str
:rtype: None
"""
```

```
def getInheritanceOrder(self):
"""
:rtype: List[str]
"""



# Your ThroneInheritance object will be instantiated and called as such:
# obj = ThroneInheritance(kingName)
# obj.birth(parentName,childName)
# obj.death(name)
# param_3 = obj.getInheritanceOrder()
```

JavaScript:

```
/**
* @param {string} kingName
*/
var ThroneInheritance = function(kingName) {

};

/**
* @param {string} parentName
* @param {string} childName
* @return {void}
*/
ThroneInheritance.prototype.birth = function(parentName, childName) {

};

/**
* @param {string} name
* @return {void}
*/
ThroneInheritance.prototype.death = function(name) {

};

/**
```

```
* @return {string[]}
*/
ThroneInheritance.prototype.getInheritanceOrder = function() {

};

/**
* Your ThroneInheritance object will be instantiated and called as such:
* var obj = new ThroneInheritance(kingName)
* obj.birth(parentName,childName)
* obj.death(name)
* var param_3 = obj.getInheritanceOrder()
*/
```

TypeScript:

```
class ThroneInheritance {
constructor(kingName: string) {

}

birth(parentName: string, childName: string): void {

}

death(name: string): void {

}

getInheritanceOrder(): string[] {

}
}

/**
* Your ThroneInheritance object will be instantiated and called as such:
* var obj = new ThroneInheritance(kingName)
* obj.birth(parentName,childName)
* obj.death(name)
* var param_3 = obj.getInheritanceOrder()
*/
```

C#:

```
public class ThroneInheritance {

public ThroneInheritance(string kingName) {

}

public void Birth(string parentName, string childName) {

}

public void Death(string name) {

}

public IList<string> GetInheritanceOrder() {

}
}

/**
* Your ThroneInheritance object will be instantiated and called as such:
* ThroneInheritance obj = new ThroneInheritance(kingName);
* obj.Birth(parentName,childName);
* obj.Death(name);
* IList<string> param_3 = obj.GetInheritanceOrder();
*/
```

C:

```
typedef struct {

} ThroneInheritance;


ThroneInheritance* throneInheritanceCreate(char* kingName) {

}
```

```
void throneInheritanceBirth(ThroneInheritance* obj, char* parentName, char* childName) {

}

void throneInheritanceDeath(ThroneInheritance* obj, char* name) {

}

char** throneInheritanceGetInheritanceOrder(ThroneInheritance* obj, int* retSize) {

}

void throneInheritanceFree(ThroneInheritance* obj) {

}

/**
* Your ThroneInheritance struct will be instantiated and called as such:
* ThroneInheritance* obj = throneInheritanceCreate(kingName);
* throneInheritanceBirth(obj, parentName, childName);

* throneInheritanceDeath(obj, name);

* char** param_3 = throneInheritanceGetInheritanceOrder(obj, retSize);

* throneInheritanceFree(obj);
*/
```

Go:

```
type ThroneInheritance struct {

}


func Constructor(kingName string) ThroneInheritance {

}
```

```
func (this *ThroneInheritance) Birth(parentName string, childName string) {

}


func (this *ThroneInheritance) Death(name string) {

}


func (this *ThroneInheritance) GetInheritanceOrder() []string {

}


/**
* Your ThroneInheritance object will be instantiated and called as such:
* obj := Constructor(kingName);
* obj.Birth(parentName,childName);
* obj.Death(name);
* param_3 := obj.GetInheritanceOrder();
*/
```

Kotlin:

```
class ThroneInheritance(kingName: String) {

fun birth(parentName: String, childName: String) {

}

fun death(name: String) {

}

fun getInheritanceOrder(): List<String> {

}

}
```

```
/**
* Your ThroneInheritance object will be instantiated and called as such:
* var obj = ThroneInheritance(kingName)
* obj.birth(parentName,childName)
* obj.death(name)
* var param_3 = obj.getInheritanceOrder()
*/
```

Swift:

```
class ThroneInheritance {

init(_ kingName: String) {

}

func birth(_ parentName: String, _ childName: String) {

}

func death(_ name: String) {

}

func getInheritanceOrder() -> [String] {

}
}

/**
* Your ThroneInheritance object will be instantiated and called as such:
* let obj = ThroneInheritance(kingName)
* obj.birth(parentName, childName)
* obj.death(name)
* let ret_3: [String] = obj.getInheritanceOrder()
*/
```

Rust:

```
struct ThroneInheritance {
```

```
}


/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl ThroneInheritance {

fn new(kingName: String) -> Self {

}

fn birth(&self, parent_name: String, child_name: String) {

}

fn death(&self, name: String) {

}

fn get_inheritance_order(&self) -> Vec<String> {

}
}

/**
* Your ThroneInheritance object will be instantiated and called as such:
* let obj = ThroneInheritance::new(kingName);
* obj.birth(parentName, childName);
* obj.death(name);
* let ret_3: Vec<String> = obj.get_inheritance_order();
*/
```

Ruby:

```
class ThroneInheritance

=begin
:type king_name: String
=end
def initialize(king_name)
```

```
end


=begin
:type parent_name: String
:type child_name: String
:rtype: Void
=end
def birth(parent_name, child_name)

end


=begin
:type name: String
:rtype: Void
=end
def death(name)

end


=begin
:rtype: String[]
=end
def get_inheritance_order()

end


end

# Your ThroneInheritance object will be instantiated and called as such:
# obj = ThroneInheritance.new(king_name)
# obj.birth(parent_name, child_name)
# obj.death(name)
# param_3 = obj.get_inheritance_order()
```

PHP:

```
class ThroneInheritance {
/**
* @param String $kingName
*/
function __construct($kingName) {

}

/**
* @param String $parentName
* @param String $childName
* @return NULL
*/
function birth($parentName, $childName) {

}

/**
* @param String $name
* @return NULL
*/
function death($name) {

}

/**
* @return String[]
*/
function getInheritanceOrder() {

}
}

/**
* Your ThroneInheritance object will be instantiated and called as such:
* $obj = ThroneInheritance($kingName);
* $obj->birth($parentName, $childName);
* $obj->death($name);
* $ret_3 = $obj->getInheritanceOrder();
*/
```

Dart:

```
class ThroneInheritance {

ThroneInheritance(String kingName) {

}

void birth(String parentName, String childName) {

}

void death(String name) {

}

List<String> getInheritanceOrder() {

}
}

/**
* Your ThroneInheritance object will be instantiated and called as such:
* ThroneInheritance obj = ThroneInheritance(kingName);
* obj.birth(parentName,childName);
* obj.death(name);
* List<String> param3 = obj.getInheritanceOrder();
*/
```

Scala:

```
class ThroneInheritance(_kingName: String) {

def birth(parentName: String, childName: String): Unit = {

}

def death(name: String): Unit = {

}

def getInheritanceOrder(): List[String] = {

}
```

```
}

/**
* Your ThroneInheritance object will be instantiated and called as such:
* val obj = new ThroneInheritance(kingName)
* obj.birth(parentName,childName)
* obj.death(name)
* val param_3 = obj.getInheritanceOrder()
*/
```

Elixir:

```
defmodule ThroneInheritance do
@spec init_(king_name :: String.t) :: any
def init_(king_name) do

end

@spec birth(parent_name :: String.t, child_name :: String.t) :: any
def birth(parent_name, child_name) do

end

@spec death(name :: String.t) :: any
def death(name) do

end

@spec get_inheritance_order() :: [String.t]
def get_inheritance_order() do

end
end

# Your functions will be called as such:
# ThroneInheritance.init_(king_name)
# ThroneInheritance.birth(parent_name, child_name)
# ThroneInheritance.death(name)
# param_3 = ThroneInheritance.get_inheritance_order()

# ThroneInheritance.init_ will be called before every test case, in which you
can do some necessary initializations.
```

Erlang:

```
-spec throne_inheritance_init_(KingName :: unicode:unicode_binary()) ->
any().
throne_inheritance_init_(KingName) ->
.

-spec throne_inheritance_birth(ParentName :: unicode:unicode_binary(),
ChildName :: unicode:unicode_binary()) -> any().
throne_inheritance_birth(ParentName, ChildName) ->
.

-spec throne_inheritance_death(Name :: unicode:unicode_binary()) -> any().
throne_inheritance_death(Name) ->
.

-spec throne_inheritance_get_inheritance_order() ->
[unicode:unicode_binary()].
throne_inheritance_get_inheritance_order() ->
.


%% Your functions will be called as such:
%% throne_inheritance_init_(KingName),
%% throne_inheritance_birth(ParentName, ChildName),
%% throne_inheritance_death(Name),
%% Param_3 = throne_inheritance_get_inheritance_order(),

%% throne_inheritance_init_ will be called before every test case, in which
you can do some necessary initializations.
```

Racket:

```
(define throne-inheritance%
(class object%
(super-new)

; king-name : string?
(init-field
king-name)

; birth : string? string? -> void?
```

```
(define/public (birth parent-name child-name)
)
; death : string? -> void?
(define/public (death name)
)
; get-inheritance-order : -> (listof string?)
(define/public (get-inheritance-order)
)))

;; Your throne-inheritance% object will be instantiated and called as such:
;; (define obj (new throne-inheritance% [king-name king-name]))
;; (send obj birth parent-name child-name)
;; (send obj death name)
;; (define param_3 (send obj get-inheritance-order))
```

Solutions

C++ Solution:

```
/*
* Problem: Throne Inheritance
* Difficulty: Medium
* Tags: string, tree, hash, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class ThroneInheritance {
public:
ThroneInheritance(string kingName) {

}

void birth(string parentName, string childName) {

}

void death(string name) {
```

```
}

vector<string> getInheritanceOrder() {

}
};

/**
* Your ThroneInheritance object will be instantiated and called as such:
* ThroneInheritance* obj = new ThroneInheritance(kingName);
* obj->birth(parentName,childName);
* obj->death(name);
* vector<string> param_3 = obj->getInheritanceOrder();
*/
```

Java Solution:

```
/**
* Problem: Throne Inheritance
* Difficulty: Medium
* Tags: string, tree, hash, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class ThroneInheritance {

public ThroneInheritance(String kingName) {

}

public void birth(String parentName, String childName) {

}

public void death(String name) {

}
```

```
public List<String> getInheritanceOrder() {

}
}

/**
* Your ThroneInheritance object will be instantiated and called as such:
* ThroneInheritance obj = new ThroneInheritance(kingName);
* obj.birth(parentName,childName);
* obj.death(name);
* List<String> param_3 = obj.getInheritanceOrder();
*/
```

Python3 Solution:

```
"""
Problem: Throne Inheritance
Difficulty: Medium
Tags: string, tree, hash, search

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class ThroneInheritance:

def __init__(self, kingName: str):


def birth(self, parentName: str, childName: str) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class ThroneInheritance(object):

def __init__(self, kingName):
"""
```

```
:type kingName: str
"""


def birth(self, parentName, childName):
"""
:type parentName: str
:type childName: str
:rtype: None
"""


def death(self, name):
"""
:type name: str
:rtype: None
"""


def getInheritanceOrder(self):
"""
:rtype: List[str]
"""




# Your ThroneInheritance object will be instantiated and called as such:
# obj = ThroneInheritance(kingName)
# obj.birth(parentName,childName)
# obj.death(name)
# param_3 = obj.getInheritanceOrder()
```

JavaScript Solution:

```
/**
* Problem: Throne Inheritance
* Difficulty: Medium
* Tags: string, tree, hash, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {string} kingName
*/
var ThroneInheritance = function(kingName) {

};

/**
* @param {string} parentName
* @param {string} childName
* @return {void}
*/
ThroneInheritance.prototype.birth = function(parentName, childName) {

};

/**
* @param {string} name
* @return {void}
*/
ThroneInheritance.prototype.death = function(name) {

};

/**
* @return {string[]}
*/
ThroneInheritance.prototype.getInheritanceOrder = function() {

};

/**
* Your ThroneInheritance object will be instantiated and called as such:
* var obj = new ThroneInheritance(kingName)
* obj.birth(parentName,childName)
* obj.death(name)
* var param_3 = obj.getInheritanceOrder()
*/
```

TypeScript Solution:

```
/**
* Problem: Throne Inheritance
* Difficulty: Medium
* Tags: string, tree, hash, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class ThroneInheritance {
constructor(kingName: string) {

}

birth(parentName: string, childName: string): void {

}

death(name: string): void {

}

getInheritanceOrder(): string[] {

}
}

/**
* Your ThroneInheritance object will be instantiated and called as such:
* var obj = new ThroneInheritance(kingName)
* obj.birth(parentName,childName)
* obj.death(name)
* var param_3 = obj.getInheritanceOrder()
*/
```

C# Solution:

```
/*
* Problem: Throne Inheritance
```

```
* Difficulty: Medium
* Tags: string, tree, hash, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class ThroneInheritance {

public ThroneInheritance(string kingName) {

}

public void Birth(string parentName, string childName) {

}

public void Death(string name) {

}

public IList<string> GetInheritanceOrder() {

}
}

/**
* Your ThroneInheritance object will be instantiated and called as such:
* ThroneInheritance obj = new ThroneInheritance(kingName);
* obj.Birth(parentName,childName);
* obj.Death(name);
* IList<string> param_3 = obj.GetInheritanceOrder();
*/
```

C Solution:

```
/*
* Problem: Throne Inheritance
* Difficulty: Medium
* Tags: string, tree, hash, search
```

```
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/




typedef struct {

} ThroneInheritance;


ThroneInheritance* throneInheritanceCreate(char* kingName) {

}

void throneInheritanceBirth(ThroneInheritance* obj, char* parentName, char*
childName) {

}

void throneInheritanceDeath(ThroneInheritance* obj, char* name) {

}

char** throneInheritanceGetInheritanceOrder(ThroneInheritance* obj, int*
retSize) {

}

void throneInheritanceFree(ThroneInheritance* obj) {

}

/**
* Your ThroneInheritance struct will be instantiated and called as such:
* ThroneInheritance* obj = throneInheritanceCreate(kingName);
* throneInheritanceBirth(obj, parentName, childName);

* throneInheritanceDeath(obj, name);
```

```
* char** param_3 = throneInheritanceGetInheritanceOrder(obj, retSize);

* throneInheritanceFree(obj);
*/
```

Go Solution:

```
// Problem: Throne Inheritance
// Difficulty: Medium
// Tags: string, tree, hash, search
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

type ThroneInheritance struct {

}


func Constructor(kingName string) ThroneInheritance {

}


func (this *ThroneInheritance) Birth(parentName string, childName string) {

}


func (this *ThroneInheritance) Death(name string) {

}


func (this *ThroneInheritance) GetInheritanceOrder() []string {

}
```

```
/**
* Your ThroneInheritance object will be instantiated and called as such:
* obj := Constructor(kingName);
* obj.Birth(parentName,childName);
* obj.Death(name);
* param_3 := obj.GetInheritanceOrder();
*/
```

Kotlin Solution:

```
class ThroneInheritance(kingName: String) {

fun birth(parentName: String, childName: String) {

}

fun death(name: String) {

}

fun getInheritanceOrder(): List<String> {

}

}

/**
* Your ThroneInheritance object will be instantiated and called as such:
* var obj = ThroneInheritance(kingName)
* obj.birth(parentName,childName)
* obj.death(name)
* var param_3 = obj.getInheritanceOrder()
*/
```

Swift Solution:

```
class ThroneInheritance {

init(_ kingName: String) {
```

```
}

func birth(_ parentName: String, _ childName: String) {

}

func death(_ name: String) {

}

func getInheritanceOrder() -> [String] {

}
}

/**
* Your ThroneInheritance object will be instantiated and called as such:
* let obj = ThroneInheritance(kingName)
* obj.birth(parentName, childName)
* obj.death(name)
* let ret_3: [String] = obj.getInheritanceOrder()
*/
```

Rust Solution:

```
// Problem: Throne Inheritance
// Difficulty: Medium
// Tags: string, tree, hash, search
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

struct ThroneInheritance {

}


/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
```

```
*/
impl ThroneInheritance {

fn new(kingName: String) -> Self {

}

fn birth(&self, parent_name: String, child_name: String) {

}

fn death(&self, name: String) {

}

fn get_inheritance_order(&self) -> Vec<String> {

}
}

/**
* Your ThroneInheritance object will be instantiated and called as such:
* let obj = ThroneInheritance::new(kingName);
* obj.birth(parentName, childName);
* obj.death(name);
* let ret_3: Vec<String> = obj.get_inheritance_order();
*/
```

Ruby Solution:

```
class ThroneInheritance

=begin
:type king_name: String
=end
def initialize(king_name)

end


=begin
```

```
:type parent_name: String
:type child_name: String
:rtype: Void
=end
def birth(parent_name, child_name)

end


=begin
:type name: String
:rtype: Void
=end
def death(name)

end


=begin
:rtype: String[]
=end
def get_inheritance_order()

end


end

# Your ThroneInheritance object will be instantiated and called as such:
# obj = ThroneInheritance.new(king_name)
# obj.birth(parent_name, child_name)
# obj.death(name)
# param_3 = obj.get_inheritance_order()
```

PHP Solution:

```
class ThroneInheritance {
/**
* @param String $kingName
*/
function __construct($kingName) {
```

```
}

/**
* @param String $parentName
* @param String $childName
* @return NULL
*/
function birth($parentName, $childName) {

}

/**
* @param String $name
* @return NULL
*/
function death($name) {

}

/**
* @return String[]
*/
function getInheritanceOrder() {

}
}

/**
* Your ThroneInheritance object will be instantiated and called as such:
* $obj = ThroneInheritance($kingName);
* $obj->birth($parentName, $childName);
* $obj->death($name);
* $ret_3 = $obj->getInheritanceOrder();
*/
```

Dart Solution:

```
class ThroneInheritance {

ThroneInheritance(String kingName) {
```

```
}

void birth(String parentName, String childName) {

}

void death(String name) {

}

List<String> getInheritanceOrder() {

}
}

/**
* Your ThroneInheritance object will be instantiated and called as such:
* ThroneInheritance obj = ThroneInheritance(kingName);
* obj.birth(parentName,childName);
* obj.death(name);
* List<String> param3 = obj.getInheritanceOrder();
*/
```

Scala Solution:

```
class ThroneInheritance(_kingName: String) {

def birth(parentName: String, childName: String): Unit = {

}

def death(name: String): Unit = {

}

def getInheritanceOrder(): List[String] = {

}

}
```

```
/**
* Your ThroneInheritance object will be instantiated and called as such:
* val obj = new ThroneInheritance(kingName)
* obj.birth(parentName,childName)
* obj.death(name)
* val param_3 = obj.getInheritanceOrder()
*/
```

Elixir Solution:

```
defmodule ThroneInheritance do
@spec init_(king_name :: String.t) :: any
def init_(king_name) do

end

@spec birth(parent_name :: String.t, child_name :: String.t) :: any
def birth(parent_name, child_name) do

end

@spec death(name :: String.t) :: any
def death(name) do

end

@spec get_inheritance_order() :: [String.t]
def get_inheritance_order() do

end
end

# Your functions will be called as such:
# ThroneInheritance.init_(king_name)
# ThroneInheritance.birth(parent_name, child_name)
# ThroneInheritance.death(name)
# param_3 = ThroneInheritance.get_inheritance_order()

# ThroneInheritance.init_ will be called before every test case, in which you
can do some necessary initializations.
```

Erlang Solution:

```
-spec throne_inheritance_init_(KingName :: unicode:unicode_binary()) ->
any().
throne_inheritance_init_(KingName) ->
.

-spec throne_inheritance_birth(ParentName :: unicode:unicode_binary(),
ChildName :: unicode:unicode_binary()) -> any().
throne_inheritance_birth(ParentName, ChildName) ->
.

-spec throne_inheritance_death(Name :: unicode:unicode_binary()) -> any().
throne_inheritance_death(Name) ->
.

-spec throne_inheritance_get_inheritance_order() ->
[unicode:unicode_binary()].
throne_inheritance_get_inheritance_order() ->
.


%% Your functions will be called as such:
%% throne_inheritance_init_(KingName),
%% throne_inheritance_birth(ParentName, ChildName),
%% throne_inheritance_death(Name),
%% Param_3 = throne_inheritance_get_inheritance_order(),

%% throne_inheritance_init_ will be called before every test case, in which
you can do some necessary initializations.
```

Racket Solution:

```
(define throne-inheritance%
(class object%
(super-new)

; king-name : string?
(init-field
king-name)

; birth : string? string? -> void?
```

```
(define/public (birth parent-name child-name)
)
; death : string? -> void?
(define/public (death name)
)
; get-inheritance-order : -> (listof string?)
(define/public (get-inheritance-order)
)))

;; Your throne-inheritance% object will be instantiated and called as such:
;; (define obj (new throne-inheritance% [king-name king-name]))
;; (send obj birth parent-name child-name)
;; (send obj death name)
;; (define param_3 (send obj get-inheritance-order))
```