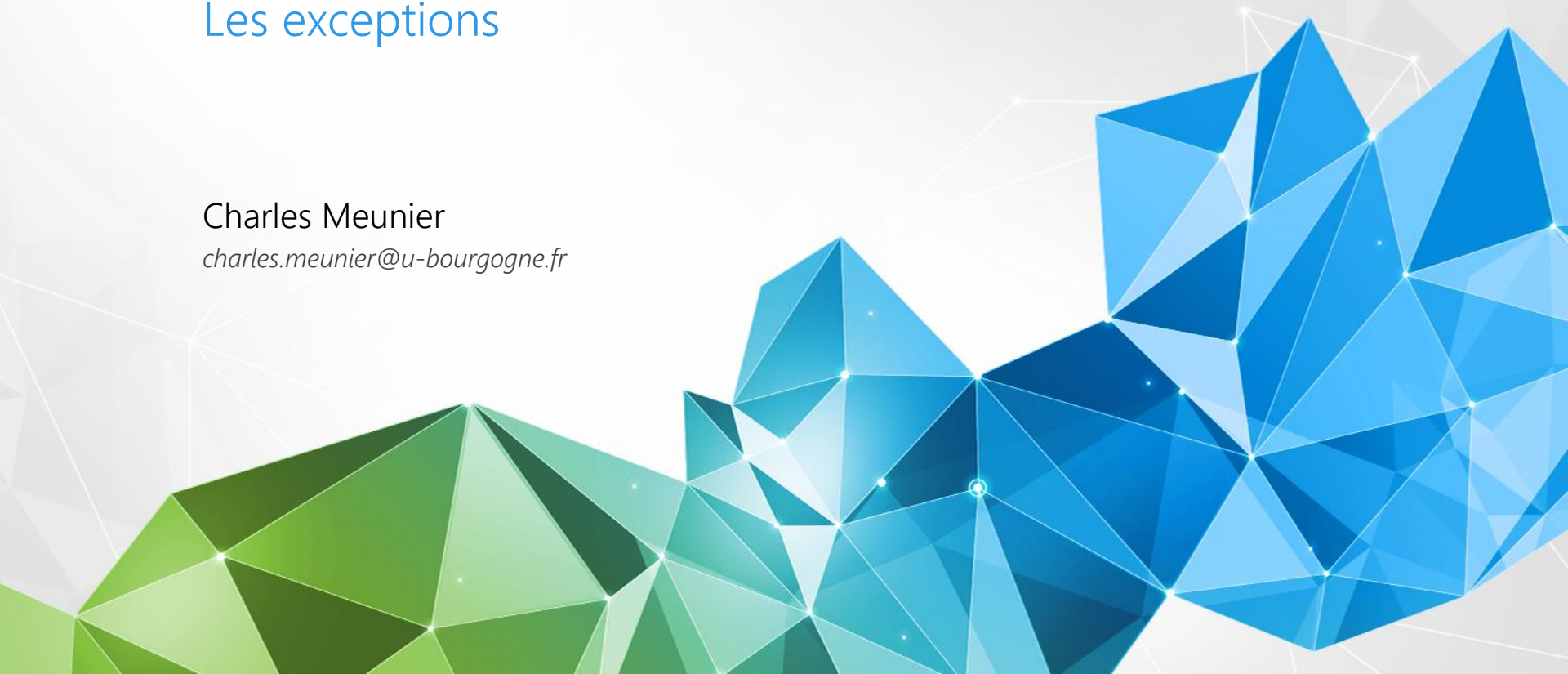
Programmation Orientée Objets

Les exceptions

Charles Meunier

charles.meunier@u-bourgogne.fr

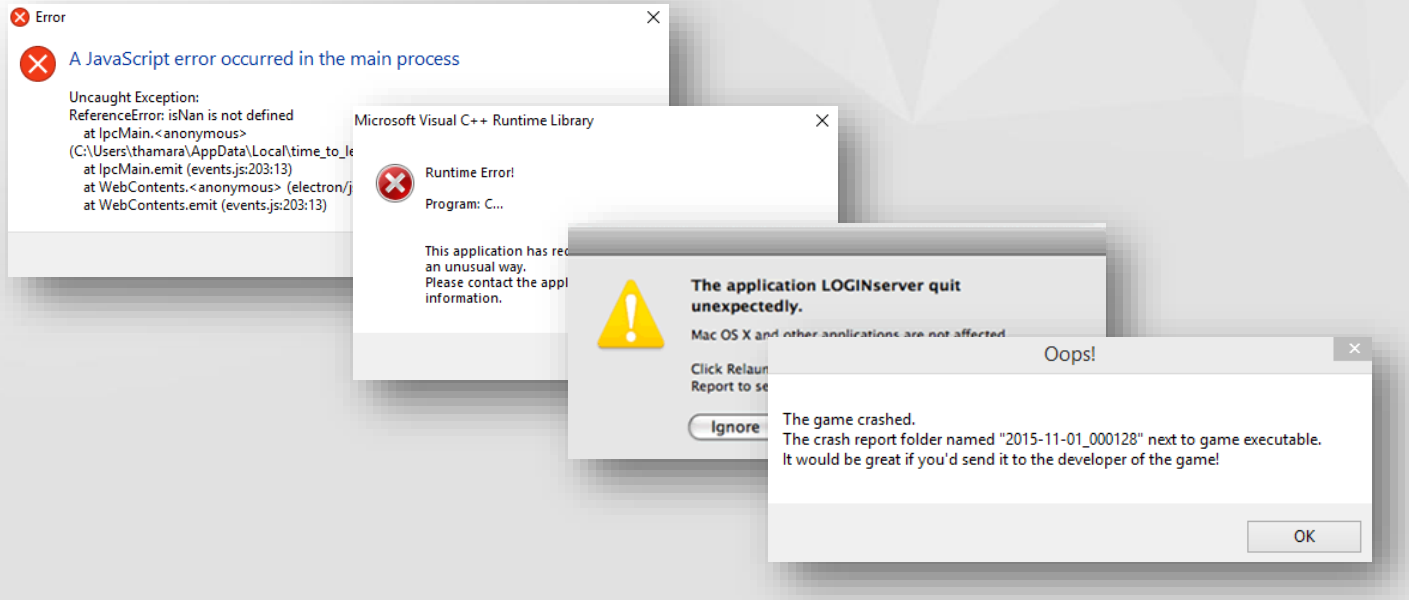


Les applications

ET LEURS ERREURS

charles.meunier@u-bourgogne.fr

ARRÊT BRUTAL DE L'APPLICATION



QUELLES CONSÉQUENCES ?

- Libération de la mémoire
- Fichier ouvert mais jamais refermé
- Crash en cours de mise à jour d'un fichier
- Service inaccessible
- …

QU'EST-CE QUI EST À L'ORIGINE DE CES ERREURS ?

- Une mauvaise saisie de l'utilisateur
- Des problèmes de droits d'écriture
- Un serveur web qui ne répond pas
- ...

UN MAUVAISE GESTION DES ERREURS PAR LE DÉVELOPPEUR

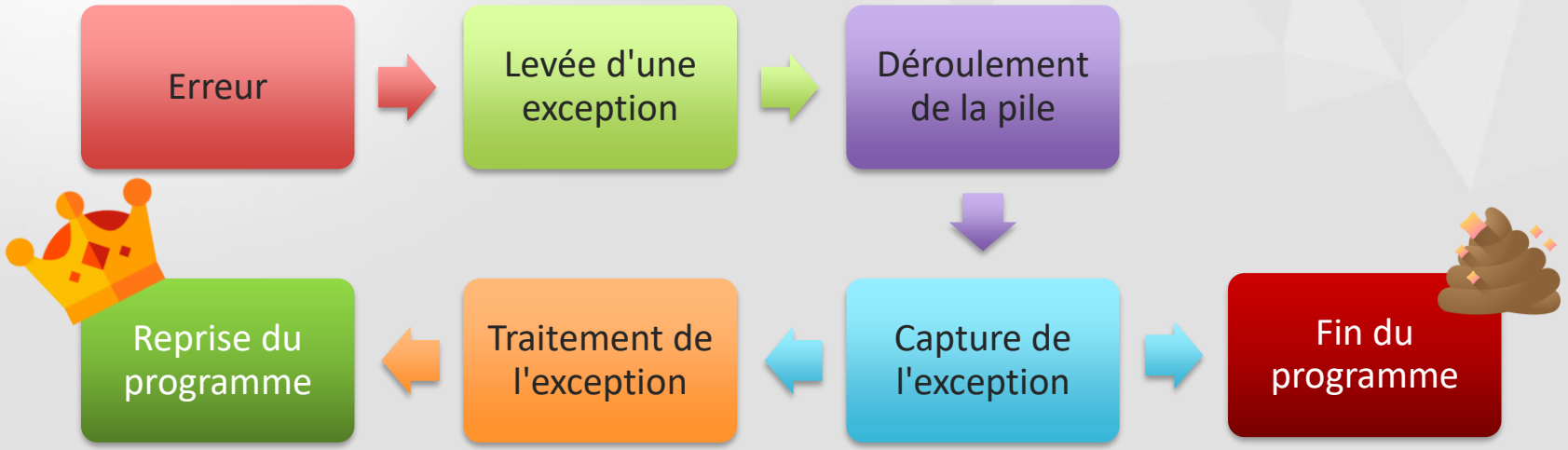
charles.meunier@u-bourgogne.fr

Gérer les erreurs avec

LES EXCEPTIONS

charles.meunier@u-bourgogne.fr

MÉCANISME D'UNE ERREUR



EXEMPLE

```
void sortirDuTableau()
{
	std::array<int, 5> tab;		// 2. creation d'un tableau de 5 éléments
	int entier = tab.at(10);	// 3. accès au 11ème élément => exception "out_of_range"
}

int main()
{
	sortirDuTableau();		// 1. Appel de la fonction sortirDuTableau

	...				// 4. suite du programme jamais exécutée

	return 0;
}
```

SUR LA PILE

```
void sortirDuTableau()
{
      std::array<int, 5> tab;
      int entier = tab.at(10);
}

int main()
{
      sortirDuTableau();

      ...

      return 0;
}
```

1 Lancement du programme > appel de main

1	Adresse de retour nécessaire pour revenir de main
2	Adresse de retour nécessaire pour revenir de sortirDuTableau
3	Sauvegarde du contexte au début de sortirDuTableau
4	Création de tab (10 x 4 octets sur la pile)
5	Création de entier (1 x 4 octets)
6	Adresse de retour nécessaire pour revenir de tab.at

FIN ANORMALE DU PROGRAMME

charles.meunier@u-bourgogne.fr

EXEMPLE AVEC CAPTURE DE L'EXCEPTION

```
void sortirDuTableau()
{
	std::array<int, 5> tab;			// 2. creation d'un tableau de 5 éléments
	int entier = tab.at(10);		// 3. accès au 11ème élément => exception "out_of_range"
}

int main()
{
	try {
		sortirDuTableau();		// 1. Appel de la fonction sortirDuTableau
	}
	catch(std::out_of_range& exception)
	{
		std::cout << "Oups !" << std::endl;
	}

	...					// 4. suite du programme

	return 0;
}
```

SUR LA PILE

```
void sortirDuTableau()
{
	std::array<int, 5> tab;
	int entier = tab.at(10);
}

int main()
{
	try {
		sortirDuTableau();
	}
	catch(std::out_of_range& exception)
	{
		std::cout << "Oups !" << std::endl;
	}

	...

	return 0;
}
```

3 4 5 6 2

1 Lancement du programme > appel de main

1	Adresse de retour nécessaire pour revenir de main
2	Adresse de retour nécessaire pour revenir de sortirDuTableau
3	Sauvegarde du contexte au début de sortirDuTableau
4	Création de tab (10 x 4 octets sur la pile)
5	Création de entier (1 x 4 octets)
6	Adresse de retour nécessaire pour revenir de tab.at

ERREUR GÉRÉE
SUITE DU PROGRAMME EXÉCUTÉE

charles.meunier@u-bourgogne.fr

TRY / CATCH

Le bloc **try** encadre une section susceptible de lever une exception.

Les blocs **catch** définissent les traitements à effectuer si une exception, dont le type est donné en paramètre, est levée.

```
try {
	// Section "dangereuse"
}
catch(TypeException_1& exception)
{
	// Traitement à réaliser
}
catch(TypeException_2& exception)
{
	// Traitement à réaliser
}
catch(...)
{
	// Traitement des autres types
	// d'exceptions
}
```

LEVER UNE EXCEPTION

- Le C++ lève très peu d'exceptions de manière native.
- Le développeur doit prévoir la levée d'exception lorsque son programme rencontre une situation anormale.
- Le mot clé **`throw`** permet de lever une exception d'un type donné.

EXEMPLE – FONCTION RACINE CARRÉE

```
float racineCarree(float valeur)
{
    return std::sqrt(valeur);
}
```

QUE FAIRE SI LA VALEUR FOURNIE VAUT -16 ?

EXEMPLE – FONCTION RACINE CARRÉE

```
float racineCarree(float valeur)
{
    if(valeur > 0)
        return std::sqrt(valeur);
    else
        return 0;
}
```

RACINE CARRÉE DE -16 RETOURNERA 0 CE QUI EST INCORRECT !

EXEMPLE – FONCTION RACINE CARRÉE

```
float racineCarree(float valeur)
{
    if(valeur > 0)
        return std::sqrt(valeur);
    else
        throw std::string("La valeur ne peut pas être négative !");
}
```

EXEMPLE – FONCTION RACINE CARRÉE

```
int main()
{
    try {
        racineCarree(-16);
    }
    catch(std::string& exception)
    {
        std::cout << "Oups !" << std::endl;
    }
}
```

Quid

DES ASSERTIONS ?

charles.meunier@u-bourgogne.fr

PRINCIPE

- Une assertion permet de tester une condition.
- Si la condition n'est pas respectée, le programme s'arrête.
- Une assertion n'est pas capturable.

```
float racineCarree(float valeur)
{
      assert(valeur > 0)

      return std::sqrt(valeur);
}
```

ASSERTION OU EXCEPTION ?

ASSERTION

- Détecte des erreurs de programmation
- Utilisée dans les tests unitaires
- N'a pas sa place en production

EXCEPTION

- Détecte les problèmes externes à la programmation
- Permet une gestion propre des erreurs
- Donne des indications claires à l'utilisateur